

Kompendium

i

COMAL

UniCOMAL-version

Martin Fahlgren -91

Innehåll

Förord	1
Vad är COMAL?	1
Varför COMAL?	1
Kompendiets uppläggning och innehåll	3
Kapitel 1	5
Tangentbordet	6
Starta COMAL-systemet	7
Lämna COMAL-systemet	8
Utskrift på bildskärmen	8
Om du skriver fel	9
Rensa skärmen	11
Infoga och utplåna text	11
Datorn utför beräkningar	12
Beräkningar med flera operationer	15
Program och programsatser	17
Tömma minnet	17
Ett litet program	17
Lista program	18
Köra program	19
Ta bort programrader	19
Ändra i program	20
Tillägg av nya programrader	21
Utskrift över hela skärmen	21
Omnumrera rader	22
Utskrift på skrivare	23
Lagra program på yttre minne	23
Hämta program från yttre minne	25
Datorn räknar	27
Om namn på variabler	28
Olika sätt att lagra värden i datorn	30
Datorn stavar	33
Dialog med datorn	36
Automatisk radnumrering	40
Datorn räknar mera	42
Mera om tilldelningssatsen	44
Användarvänlig utskrift	45
Stränghantering	50
COMal ihåg	55
Kapitel 2	57
Inledning	57
Programexemplen	58
Kapitel 3	69
Rättelser	69
Mera om villkorssatser	71
Boolska datatyper - TRUE/FALSE	73
Operatorerna DIV och MOD	76
Mera om upprepningssatser - LOOP-ENDLOOP	79
Sökning	80
Kapitel 4	81
Rättelser	81
Boolska funktioner	85
Underprogram - en sammanfattning	86
Procedurer	86
Parameteröverföring	86

Slutna procedurer	87
Referensanrop	87
Indexerade variabler	88
Funktioner	88
Avslutande kommentarer	90
Kapitel 5	91
Rättelser	91
Filhantering	93
Olika filtyper	93
Sekventiella filer	93
Direktfiler	95
Övningsuppgifter	96
Appendix 1 - För den som vill veta mer	101
Mera om SELECT OUTPUT	101
Tidmätning	101
Sortering	101
Binärsökning	102
Felhantering	103
Färger m m	104
Grafik	105
Hexadecimala och binära tal	105
Appendix 2	107
Systemkommandon	107
Editering m m	110
Funktionstangenter	112
Lösningar	113
ASCII-tabell (IBM-ASCII)	121
Sakregister	123

Förord

Vad är COMAL?

COMAL är en modern s k strukturerad variant av programspråket BASIC. Det har sitt ursprung i Danmark på 70-talet. De två upphovsmännen, Børge R. Christensen och Benedict Löfstedt, hade det uttalade syftet att skapa ett programspråk som kombinerade enkelheten i traditionell BASIC med kraftfullheten i Pascal (särskilt dess programstruktur). Sedan dess har även andra ”strukturerade” BASIC-dialekter, med programspråkskonstruktioner liknande dem i COMAL, sett dagens ljus (såsom QuickBASIC och TRUE BASIC).

Varför COMAL?

De moderna strukturerade BASIC-dialekterna, dit COMAL hör, har framtagits därför att mycken befogad kritik riktats mot de gamla, mer eller mindre primitiva BASIC-dialekterna (som varit vanliga på s k hemdatorer):

- Programmen blir svårlästa (kryptiska) och därmed svåra att förstå, rätta (”avlusa”) och ändra.
- Många viktiga programmeringsproblem är mycket svåra (vissa nästan omöjliga) att lösa.
- Det är svårt att lära sig och lära ut goda programmeringsvanor.

De ”ostrukturerade” BASIC-dialekterna är av dessa anledningar (och ytterligare några) olämpliga som nybörjarspråk. Risken är stor att de som lär sig att ”programmera” med sådana språk, särskilt om det sker utan sakkunnig ledning, helt enkelt blir dåliga programmerare. Och inte nog med det: Det är svårt att ”omskola” dem som lagt sig till med de ovanor som primitiva BASIC-dialekter inbjuder till. Det har gått så långt att representanter för databranschen uttryckt att det är bättre att en nyanställd inte kan programmera alls än att denne lärt sig ”programmera” vanlig (läs: ”dålig”) BASIC.

Ovan skrevs ordet programmera inom citationstecken. I själva verket är det ofta fel att tala om programmering i sådana fall. Det är fråga om ”kodning”. *Programmering* innebär något mer än att prestera ett mer eller mindre väl fungerande program i något programspråk, vilket förhoppningsvis denna kurs kommer att klargöra.

På grund av den traditionella BASICens brister har man i den grundläggande programmeringsutbildningen på våra högskolor under lång tid använt andra programspråk (tidigare främst Pascal och numera även Ada, Modula-2 och PROLOG). Problemet med Pascal och övriga ”högskolespråk” är att de, jämfört med BASIC, kräver mer innan man ”kommer till skott”. När det gäller Ada saknas dessutom ”användarvänliga” utvecklingsmiljöer på persondatorer, vilka är vanligast på våra gymnasier och vuxengymnasier.

COMAL är en ”kompromiss” med flera fördelar. Det är lika enkelt att använda som traditionell BASIC (eller enklare), samtidigt som det är ”strukturerat” och ”modulärt”. COMAL understödjer inläring av goda programmeringsvanor och motverkar det ”ostrukturerade” och ologiska tänkande som traditionell BASIC inbjuder till. COMAL har många av de ”språkelement” som behövs för att man på ett naturligt sätt ska kunna lära sig att *programmera* – och detta är viktigare än att lära sig *koda* i någon mer eller mindre ”populär” programspråksdialekt, ty programspråk nyskapas, utvecklas och förändras ständigt, medan goda programmeringsprinciper och -metoder alltid är bra att ha i bagaget och i stort sett oberoende av programspråk.

När du genomfört denna kurs kommer du (förhoppningsvis) att ha lärt dig

- koda i ett kraftfullt, modernt programspråk
- goda programmeringsvanor
- grunder som underlättar inlärandet av andra moderna programspråk, som Pascal, Modula-2, C och Ada.

Dessa är de viktigaste anledningarna till att vi använder COMAL som nybörjarspråk, men det finns fler:

- Bra COMAL-versioner ("interpretatorer") finns till både persondatorer (IBM och "kompatibler", Amiga och Macintosh) och större datorer.
- På svenska finns en relativt riklig COMAL-litteratur, vilket hänger samman med att skillnaderna mellan olika COMAL-versioner är ganska små (COMAL är ganska "standardiserat"), detta i motsats till andra BASIC-dialekter, där floran är enorm.
- COMAL är ett vanligt skolspråk, varför många läroböcker i andra ämnen (matematik, fysik osv) ger programexempel skrivna i COMAL.

Kompendiets uppläggning och innehåll

I kompendiet ingår ändringar, kommentarer och tillägg till Olsson: *GRUNDLÄGGANDE PROGRAMMERING I COMAL*.

Olssons bok är skriven för Compis-COMAL, som i vissa avseenden skiljer sig från den COMAL-version vi har (UniCOMAL). Kompendiets främsta målsättning är att bena ut dessa problem, men det innehåller också annat, kompletterande material.

De flesta skillnaderna mellan olika COMAL-versioner rör den allmänna ”miljö” som man jobbar i (start av COMAL-systemet, redigering av program m m), medan de språkliga skillnaderna är relativt små. Lärobokens första kapitel, som till stor del rör frågeställningar av första slaget, har därför helt omskrivits. Kapitel 1 i kompendiet *ersätter* lärobokens 1:a kapitel och kan läsas helt fristående från läroboken.

Lärobokens kapitel 2, som rör grafik, ges en mer styvmoderlig behandling: Programexemplen har ombearbetats så att de kan köras med UniCOMAL. Grafikhantering skiljer sig en hel del mellan Compis-COMAL och UniCOMAL, varför kapitlet egentligen skulle behöva en grundligare ombearbetning än kapitel 1. Detta är nu inget som är specifikt för olika COMAL-versioner, utan grafik är överhuvudtaget dåligt standardiserad. Av dessa och andra skäl brukar inte grafik ingå i grundläggande programmeringskurser. Grafik brukar dock väcka intresse hos många elever och förhoppningen är att kompendiet tillsammans med lärobokskapitlet ska fungera som en introduktion till ämnet. För att tränga in djupare i grafikprogrammeringens alla hemligheter, hänvisas till manualer och speciallitteratur.

De resterande lärobokskapitlen (3-5) tas upp kapitelvis, med 2 separata avdelningar för varje kapitel:

- En genomgång sida för sida av lärobokstexten, där de detaljer som skiljer UniCOMAL från Compis-COMAL påpekas och justeras. Dessutom rättas några felaktigheter. *Titta alltid igenom dessa sidor innan du läser motsvarande lärobokskapitel.*
- Kompletteringar och tillägg till läroboken (operatorerna DIV och MOD, de boolska konstanterna TRUE/FALSE, en introduktion till text- och direktfiler m m). Även kompletterande övningsuppgifter ingår. Förslag på var avsnitten kan inpassas i lärobokstexten ges i anslutning till texterna. Vilka avsnitt som bör medtas beror givetvis på kursens omfattning och inriktning. De avsnitt som förbigås kan förhoppningsvis ändå vara till nytta och glädje för intresserade elever.

NOTERINGAR

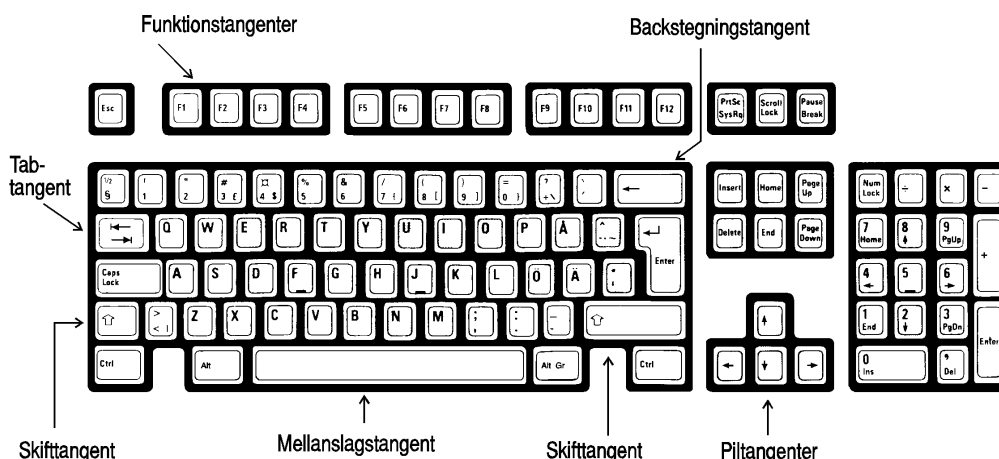
Kapitel 1

I detta kapitel får du lära dig

- starta och lämna COMAL-systemet
- använda den texteditor som finns inbyggd i COMAL
- COMAL-kommandon för att
 - lista program (LIST)
 - köra program (RUN)
 - lagra (skriva) program på yttre minne (SAVE)
 - hämta (läsa) program från yttre minne (LOAD)
- grundläggande datatyper: tal- och strängvariabler
- användarvänlig inmatning (indata) och utmatning (utdata)
- enkla beräkningar och lite texthantering med datorn

Om du tidigare jobbat med någon variant av BASIC kommer mycket i detta kapitel att tjäna som repetition. Även om du tycker att du behärskar grunderna i BASIC är det klokt att genomarbeta texten och åtminstone en del av övningarna, ty med all säkerhet finns avvikelser från den BASIC som du är van vid. En del viktiga programmeringsprinciper introduceras också. Dessutom kommer du att upptäcka en hel del finesser som förmodligen saknas i den BASIC-dialekt du eventuellt träffat på tidigare.

Tangentbordet



Bilden ovan visar ett tangentbord till en modern PC (s k PS/2). Ditt tangentbord kan ha en något annorlunda utformning, t ex är funktionstangenterna placerade längst till vänster på äldre tangentbord, men i det stora hela bör det se ut som i figuren.

De flesta tangenterna på den vänstra, större delen är bokstavs- och sifvertangenter. De brukar med ett gemensamt namn kallas för *alfanumeriska* tangenter. Övriga tangenter på denna del av tangentbordet har följande funktioner (se figuren):

Mellanslagstangent. Ger en tom position (kallas mellanslag eller blanktecken).

Enter-tangent (inmatningstangent, ibland märkt **Return** och/eller med en vinkelpil, ↵). Har ungefär samma funktion som vagnreturen på en skrivmaskin. Används för att avsluta inmatning av en rad. I fortsättningen betecknas denna tangent med symbolen ↵.

Skifttangent. Med denna nedtryckt ändras tangentfunktionerna så att stora bokstäver resp tecknen som står överst till vänster på tangenterna erhålls. När den släpps återgår skriften till det ”normala”. OBS att det finns två skifttangenter, en på vardera sidan om tangentbordet.

Ctrl-tangent (kontrolltangent). Används tillsammans med andra tangenter ungefär som skift-tangenten, dvs den ändrar på de ”vanliga” tangenternas funktion. När **Ctrl**-tangenten ska nedtryckas tillsammans med en annan tangent används ofta skrivsättet **Ctrl-A**, vilket betyder att **Ctrl** ska hållas nedtryckt när **A**-tangenten trycks. Nyare tangentbord brukar ha två **Ctrl**-tangenter.

Alt-tangent. Används i kombination med andra tangenter ungefär som **Ctrl**-tangenten. Nyare tangentbord brukar ha två **Alt**-tangenter.

Caps Lock-tangenten (”skiftlås”). Behandlas senare.

Backstegningstangent (”**BackSpace**”). Flyttar markören ett steg åt vänster och raderar det tecken som passeras.

Esc-tangent. Används för att avbryta vissa operationer. Denna tangent kommer vi inte att utnyttja så mycket.

Tab-tangent. Flyttar markören till nästa tabulatorstopp.

Starta COMAL-systemet

Starta COMAL

Hur detta går till beror delvis på datortypen och den utrustning som datorn är försedd med.

Dator med hårddisk:

Slå på dator och bildskärm. När datorn ”kommit igång” och är redo att ta emot ”kommandon” är det för det mesta mycket enkelt att starta COMAL-systemet, men det kan skilja sig något mellan olika maskiner. Din lärare ger dig nödvändig information.

Dator utan hårddisk:

För att köra COMAL behöver du en diskett med UniCOMAL. Denna får du av din lärare.

Gör följande:

- 1) Slå på dator och bildskärm.
- 2) Stoppa COMAL-disketten i vänster skivenhet (A:-enheten) och stäng luckan till skrivstationen (efter en liten stund börjar datorn att arbeta).
- 3) Svara ev på datorns frågor om datum och tid.
- 4) Skriv **CO** och tryck ↵, dvs inmatningstangenten (datorn läser därefter in COMAL i arbetsminnet).

När COMAL-systemet startat och är klart för användning ser bildskärmen ut ungefär så här:

```

The UniComal Personal Computer COMAL
Version 2.20 using Math Emulation.
(C) Copyright UniComal A/S 1989.
    All rights reserved.
Registration number: 8901-26371

1LIST 2RUN+ 3AUTO+ 4RENUM+ 5DIR+ 6EDIT 7CON+ 8LOAD " 9FIND " 0SEL CON

```

Skulle något gå snett, så att du t ex hamnar i något annat program, eller om datorn ”låser sig”, då kan du göra omstart genom att trycka de 3 tangenterna **Ctrl-Alt-Del** (tryck i denna ordning och håll dem nedtryckta – om datorn är försedd med hårddisk, ska i så fall luckan till flexskivenheten vara öppen).

Lämna COMAL-systemet

Ibland behöver man lämna COMAL-systemet. Detta är enkelt: Skriv BYE och tryck ↵. Observera att COMAL-systemet då måste återstartas om du vill komma in i COMAL på nytt.

Utskrift på bildskärmen

Vi förutsätter nu att COMAL-systemet är igång.

På bildskärmen ser du en blinkande fyrkant. Denna kallas för *markör* (engelska: *cursor*) och visar var nästa tecken som skrivs in från tangentbordet kommer att hamna.

Skriv på tangentbordet

```
PRINT "Det är lätt att lära sig COMAL"
```

och tryck ↵. Glöm inte citationstecknen – i synnerhet inte det första.

För att skriva en stor bokstav, håll någon av de två skifttangenter nedtryckt när du skriver bokstaven. För att skriva flera stora bokstäver efter varandra, kan du först trycka **Caps Lock** (skiftlåset), varvid stora bokstäver fås utan att någon skifttangenter hålls nedtryckt. Ett nytt tryck på **Caps Lock** stänger av skiftlåset. Använder du en skifttangenter när skiftlåset är påslaget fås små bokstäver. Lagg märke till att tangentbord ofta har en liten lampa som är tänd när skiftlåset är aktivt.

Här i texten skriver vi COMAL-ord, såsom ordet PRINT, med stora bokstäver, men du kan lika gärna använda små, ty datorn ”gör om” alla bokstäver i COMAL-ord till stora. Det är därför vi skriver COMAL-orden med stora bokstäver.

På bildskärmen står nu följande (om du gjort rätt):

```
PRINT "Det är lätt att lära sig COMAL"
Det är lätt att lära sig COMAL
```

Den rad du skrivit innehåller ett *kommando*, **PRINT** (betyder ”skriv ut”) och en text (det som står inom citationstecken). Du har med ordet PRINT beordrat datorn att skriva texten. I datorsammanhang kallas en sådan text för en *sträng*. En sträng kan innehålla bokstäver, siffror och andra tecken, t ex kommatecken och mellanslag (blanktecken).

Med ordet PRINT kan du få datorn att skriva ut vilken text som helst. Skriver du

```
PRINT "Hejsan på dejsan!"
```

och trycker ↵, svarar datorn

```
Hejsan på dejsan!
```

Skriver du

```
PRINT "      här"
```

(ordet föregås av 5 blanktecken) så svarar den

```
      här
```

Blanktecknen skrivs också ut.

Exempel

Antag att du mitt i en rad upptäcker att du skrivit fel:

```
PRINT "CONAL, ett■
```

Detta kan rättas till så här

- Flytta markören åt vänster genom att upprepade gånger trycka pil-vänster (←) tills markören står på N i CONAL. Du kan även få tangenten repeterande genom att hålla den nedtryckt
- Skriv rätt bokstav, M
- Flytta tangenten tillbaka till platsen där du nyss slutade skriva och fullborda raden (du kan trycka på **End** för att direkt flytta till radens slut)

```
PRINT "COMAL, ett programspråk för grundutbildning"
```

För att COMAL-systemet ska utföra/registrera det du skrivit, måste du alltid trycka ↵ när en rad är färdigskriven. Detta låter vi vara underförstått i fortsättningen.

Observera att

```
PRINT "CONAL, ett programspråk för grundutbildning"
```

skulle ge utskriften

```
CONAL, ett programspråk för grundutbildning
```

Även om du stavat fel, skriver datorn ut precis det du skrivit mellan citationstecknen.

Däremot får du inte skriva fel på ett COMAL-ord:

```
PTINT "Blaha"
```

Datorn förstår inte! När du trycker på ↵ vägrar den att lämna raden och du får ett felmeddelande under den felaktiga raden. I detta fall skriver den

```
":=" eller "(" förväntat, ej textkonstant
```

På detta stadium förstår du nog inte vad datorn menar, men detta att du får ett felmeddelande betyder att något är fel: Du har ej följt de regler som gäller i COMAL. Här är det väl heller inte så svårt att reda ut vari det verkliga felet består?

Rätta felet genom att ändra

```
PTINT till PRINT
```

Tryck sedan ↵. Du behöver inte befinna dig i slutet av raden när du gör detta. Datorn ”läser” alltid av hela raden där markören står.

Ett annat sätt att rätta fel är att lämna raden och börja om. Det kan du göra antingen genom att byta rad med pil-ner-tangenten (↓) eller genom att samtidigt trycka på tangenterna märkta **Ctrl** och **Break** (den sistnämnda ligger uppe till höger på tangentbordet).

Studera alltid datorns felmeddelanden. Ofta ger de en direkt fingervisning om vari felet består. Du lär dig då också att allt bättre ”tolka” felmeddelanden, dvs snabbt inse vad som är fel.

Övning 1.4

a) Skriv

```
PRINT "Mikrodatorn i skolan"
```

Vad skriver datorn ut?

b) Skriv

```
PRINS "Datorn gör ej som jag vill"
```

Vad skriver datorn ut?

Övning 1.5

a) Skriv (ordagrant)

```
PRINT "COMAL, ett programspråk för grundutbildn■"
```

Rätta raden så att den blir

```
PRINT "COMAL, ett programspråk för grundutbildning"
```

Tips: Om du vill förflytta dig till början av en lång rad kan du hoppa direkt dit med en tryckning på **Home**-tangentsen och efter rättningen återgå till radslutet med tryckning på **End**.

Rensa skärmen

När du skrivit en stund fylls bildskärmen med text. Det gör nu ingenting, ty när du kommer till skärmens nedersta rad flyttas texten automatiskt uppåt ett steg för varje ny rad.

Skulle du vilja rensa bort all gammal text från skärmen går det också fint: Håll **Ctrl**-tangentsen nedtryckt och tryck på **Home**, vilket vi med ett förkortat skrivsätt skriver **Ctrl-Home**. Då raderas skärmen och markören flyttas till första skärmraden.

Infoga och utplåna text

Det händer ofta att man vill ändra, lägga till och ta bort text inuti en redan skriven rad.

Exempel

Antag att någon använt datorn och skrivit

```
PRINT "Brage är Sveriges bästa fotbollslag"
```

och att en göteborgare, hammarbyit, ... får se detta. Han/hon vill givetvis snabbt ändra:

- Flytta markören med piltangenterna till B i Brage (även pil-upp och pil-ner kan användas).
- Radera texten Brage med tryck på tangentsen märkt **Del**.
- Tryck på **Ins**-tangentsen. Då övergår datorn i infogningsläge och vi kan skriva in ny text utan att skriva över den gamla. Skriv t ex Blåvitt och tryck ↵ för att få nya texten utskriven.

Ett annat (snabbare) sätt att utföra ändringen är att först skriva över den texten som ska ändras (Brage) och sedan (om så behövs) trycka på **Ins** och infoga resten.

Tryck på **Ins** ett par gånger och iaktta hur markören ändras:

- I överskrivningsläge blinkar markören.
- I infogningsläge utgör markören ett fast block (blinkar ej).

För att utplåna text till vänster om markören, använder man bakåtstegningstangenten (**Back Space**). Texten till höger om markören följer då med åt vänster. Vill man sedan infoga text trycker man först på **Ins**. För att åter skriva över gammal text trycker man **Ins** igen.

För att utplåna all text till *höger* om markören kan du trycka

Ctrl-End

dvs **Ctrl**-tangenten och tangenten märkt **End**. Om markören då står längst till vänster på raden utplånas hela raden.

Övning 1.6

Använd redigeringstangenterna i följande övningar.

a) Skriv först följande text:

```
PRINT "Man skall inte sälja skrinet, förrän björnen är sjuk"
```

Rätta detta till

```
PRINT "Man skall inte sälja skinnet, förrän björnen är skjuten"
```

b) Ändra raden

```
PRINT "Stock är stad i Norge."
```

till

```
PRINT "Stockholm är huvudstad i Sverige."
```

Datorn utför beräkningar

I datorns barndom utnyttjades datorer främst för beräkningar. I dag har datorer betydligt mångsidigare användningsområden, men det faktum att de snabbt och felfritt kan utföra omfattande beräkningar förblir en av de viktigaste egenskaperna.

Skriv

```
PRINT "Svar = 5"
```

Datorn svarar

```
Svar = 5
```

Skriv i stället

```
PRINT "Svar =" ; 5
```

Datorn replikerar

Svar = 5

Det blir ingen skillnad, men du misstänker förmodligen att det måste vara det eftersom uttrycket efter ordet PRINT inte är detsamma. Låt oss pröva igen!

Skriv

```
PRINT "Summan = 3+5"
```

Datorn svarar

Summan = 3+5

Skriv

```
PRINT "Summan =" ; 3+5
```

Datorn svarar nu

Summan = 8

Denna gång ser vi skillnaden! Datorn har räknat ut 3+5 och skrivit ut resultatet. Ordet PRINT kan således användas både för att

- skriva ut text (mellan citationstecken)
- beräkna och skriva ut värdet av beräkningsuttryck

Ett *semikolon* (;) ska finnas mellan strängen och uttrycket som ska beräknas. I vissa situationer används kommatecken i stället, vilket vi återkommer till längre fram.

Exempel

Skriv

```
PRINT "Differensen =" ; 1.0-1.0
```

Datorn svarar

Differensen = 0

Lägg märke till att punkt används som decimaltecken, precis som på miniräknare.

Exempel

Skriv

```
PRINT "Produkten =" ; 4*7
```

Datorn svarar

Produkten = 28

Lägg märke till att multiplikationstecken skrivs med *.

Exempel

Skriv

```
PRINT "Kvoten =" ; 1.414 / 2
```

Datorn svarar

```
Kvoten = 0.707
```

Lägg märke till att snett bråkstreck (/) används som divisionstecken.

Exempel

Skriv

```
PRINT "Potensen =" ; 6.5 ^ 2
```

Datorn svarar

```
Potensen = 42.25
```

Lägg märke till att ”upphöjt till” skrivs med taksymbolen (^).

I ”matten” skriver man som bekant $6,5^2$.

De fem symbolerna (räknetecknen) +, -, *, / och ^ brukar kallas för *aritmetiska operatorer* (ordet aritmetisk kommer från ett grekiskt ord som betyder räkna).

Av exemplen ovan förstår du nog att datorn kan användas ungefär som en miniräknare. Ordet PRINT kan då sägas motsvara tangenten märkt = på miniräknaren. Viktiga skillnader mellan en vanlig miniräknare och datorn är dock att vi i senare fallet både kan bifoga text och redigera inmatningen.

Övning 1.7

Använd kommandot PRINT för att, som i exemplen ovan, skriva ut texten och beräkna uttrycket

- a) Summan = $47+2$
- b) Produkten = $47*2$
- c) Summan = $497+2$
- d) Produkten = $497*2$
- e) Differensen = $3-0.75$
- f) Produkten = $3*0.75$
- g) Differensen = $7-0.875$
- h) Produkten = $7*0.875$

Siffran 0 kan utelämnas före decimaltecknet. Man kan t ex skriva .875 i stället för 0.875.

Övning 1.8

Använd PRINT-kommandot för att skriva ut texten och beräkna uttrycket

- a) Kvoten = $4181/2584$
- b) Summan = $2584+4181$
- c) Kvoten = $6765/4181$

Övning 1.9

Använd kommandot `PRINT` för att skriva ut texten och beräkna uttrycket

- a) Kvadraten = $1.414121 * 1.414121$
- b) Kvadraten = 1.414121^2
- c) Potensen = 2^{16}
- d) Potensen = 1.258925^{10}

Beräkningar med flera operationer

Nu ska vi undersöka hur datorn behandlar sammansatta räkneuttryck, s k *aritmetiska uttryck*.

Skriv

```
PRINT 3+5/5
```

Ge akt på datorns svar. Skriv sedan

```
PRINT (3+5)/5
```

Jämför med resultatet du fick nyss. Varför erhålls olika resultat?

Skriv nu

```
PRINT 3*3/5
```

Och därefter

```
PRINT 3*(3/5)
```

Jämför svaren. Är det någon skillnad?

Låt oss också pröva potenser. Skriv

```
PRINT 3+5^3
```

och

```
PRINT (3+5)^3
```

Jämför svaren.

Som framgår av resultaten ovan utförs beräkningarna i en viss ordning när flera räkneoperationer ingår. Man säger att räkneoperationerna har olika *prioritet*.

Prioritetsreglerna överensstämmer med de som gäller i matematiken:

- först utförs potenser, ”upphöjt till” (^)
- sedan utförs multiplikation och division (* resp /)
- sist utförs addition och subtraktion (+ och -)

Detta kan även uttryckas: Av de aritmetiska operatorerna har ^ högst och +, - lägst prioritet.

Om operatörer med samma prioritet (multiplikation och division eller addition och subtraktion) ingår i ett uttryck utförs de i tur och ordning, från vänster till höger.

Ordningen, prioriteten i beräkningen kan ändras med parenteser: Parentesuttryck beräknas först!

Övning 1.10

Använd kommandot PRINT för att beräkna uttrycken

- a) $1.3 - 0.3^3$
- b) $(1.3 - 0.3)^3$
- c) $456456/13/11/7$
- d) $456456/(13*11*7)$

Övning 1.11

Använd kommandot PRINT för att beräkna uttrycken

- a) $1/(2-1.9)$
- b) $1/2-1.9$
- c) $2^{10}-1$
- d) $2^{(10-1)}$

Övning 1.12

Använd kommandot PRINT för att beräkna

- a) $0.2346*426.26$
- b) $23.46/100*426.26$
- c) $1.2346*809.98$
- d) $(1+23.46/100)*809.98$

Övning 1.13

a) Använd kommandot PRINT för att skriva ut texten nedan och beräkna uttrycket

Moms på 6800 kr = $25/100*6800$ kr

eller

Moms på 6800 kr = $0.25*6800$ kr

b) Använd kommandot PRINT för att skriva ut texten nedan och beräkna uttrycket

Pris inkl. moms = $2400+0.25*2400$ kr

eller

Pris inkl. moms = $1.25*2400$ kr

Övning 1.14

Använd PRINT-kommandot för att skriva ut texten nedan och beräkna uttrycket

12% ränta på 45000 = $(12/100)*45000$

Program och programsatser

Om du skriver

```
PRINT "Datorer"
```

och trycker ↵, skrivs det som står mellan citationstecknen ögonblickligen ut. Detta sätt att använda datorn brukar kallas för att arbeta i *direktmod*. Vi har hittills bara arbetat på detta enkla sätt: Vi har beordrat datorn att omedelbart skriva ut text och/eller utföra någon beräkning. Oftast vill vi dock att datorn skall utföra mer komplicerade saker. För detta behövs ett *program*.

Skriv nu i stället

```
10 PRINT "Datorer"
```

och tryck ↵

Under förutsättning att du inte skrivit något fel erhålls denna gång ingen utskrift alls, utan markören flyttas bara ned till nästa skärmrad och väntar på ny information.

Genom att skriva ett radnummer längst till vänster på varje rad kan vi lagra innehållet i raden i datorns minne. Radnumret ska vara ett heltal mellan 1 och 9999. Om man skriver ett radnummer som ligger utanför detta intervall, fås felmeddelandet:

```
Radnumren går från 1 till 9999
```

Innehållet i en numrerad rad kallas för en *programrad*. Ett COMAL-program består av sådana programrader. En programrad innehåller normalt en enda utförbar *programsats*, t ex en PRINT-sats.

Datorn kontrollerar alltid att programsatser följer de språkregler som finns i COMAL. Vid brott mot någon sådan regel ger datorn ett felmeddelande, som anger vilken regel vi brutit mot.

Tömma minnet

När man ska börja inmata ett program kan det hända att det redan finns ett program i datorns minne. För att inte de gamla programraderna ska "blandas" med de nya, bör det gamla programmet raderas. Det görs med kommandot **NEW**. Skriv

```
NEW
```

följt av en tryckning på ↵.

Som påpekats tidigare kan vi också skriva nyckelordet **new**, liksom alla andra COMAL-ord, med små bokstäver.

Ett litet program

Skriv

```
10 PRINT "Djur i skogen:"
```

När vi trycker på ↵ flyttas markören ned till nästa rad, vilket betyder att satsen är korrekt och därmed lagrad i datorns minne.

Vi kan nu skriva några satser till:

```
20 PRINT "Myra"
30 PRINT "Räv"
40 PRINT "Älg"
```

Det är ofta praktiskt att numrera raderna 10, 20, 30..., eftersom det kan hända att man senare vill lägga in nya rader mellan de man redan inmatat.

Lista program

För att få en utskrift av programraderna i datorns minne kan man skriva **LIST** (och trycka ↵). **LIST** måste stå ensamt på raden. Det gäller för övrigt alla kommandon. Det får alltså inte finnas någon annan text på raden där du skriver **LIST**. Om det inte finns någon tom skärmrad kan du antingen rensa raden efter ordet **LIST** genom att trycka **Ctrl-End**, eller trycka pil-ner-tangenten tills dess en tomrad ”rullat upp”.

Skriv

```
LIST
```

Datorn svarar med att ge följande utskrift på bildskärmen:

```
0010 PRINT "Djur i skogen:"
0020 PRINT "Myra"
0030 PRINT "Räv"
0040 PRINT "Älg"
```

LIST skriver ut alla rader i minnet, men man kan också på olika sätt begränsa utskriften:

```
LIST 30
```

skriver enbart ut rad 30.

```
LIST 30-100
```

skriver ut raderna f o m 30 t o m 100 (om några sådana finns).

```
LIST 50-
```

ger utskrift av raderna f o m rad 50 till programmets slut.

```
LIST -200
```

skriver ut alla rader från programmets början t o m rad 200.

En programlistning kan stoppas och återstartas med tryck på mellanslagstangenten, vilket är bekvämt då man vill titta på längre programavsnitt.

Köra program

För att få programsatserna i ett program utförda måste vi ge ett speciellt kommando: **RUN**. Man säger att programmet *körs* (eller *exekveras*) med **RUN**-kommandot.

Kommandon får inte föregås av radnummer. Skriv därför **RUN** på en tom skärmrad och tryck ↵.

Vårt program innehåller enbart **PRINT**-satser och ger följande utskrift på bildskärmen:

```
Djur i skogen:
Myra
Räv
Älg
```

Programmet avslutat

Den sista raden lägger datorn till för att meddela att den är klar med programkörningen.

Funktionstangenterna

Som du säkert lagt märke till är skärmens nedersta rad fylld med text när du arbetar med COMAL. Denna rad visar funktionen hos de s k *funktionstangenterna*, märkta **F1-F10**. På lite äldre datorer är dessa tangenter placerade längst till vänster på tangentbordet och på nyare modeller ligger de på en egen rad längst upp.

Tryck på funktionstangenter ger det resultat som framgår av alternativen på nedersta raden. Ett tryck på **F2** ger t ex samma effekt som om du skriver **RUN** och sedan trycker ↵.

Funktionstangenterna kan bespara en hel del tangenttryckande.

Lägg också märke till att om du håller *skift*- eller *Ctrl*-tangenter nedtryckt erhålls ytterligare uppsättningar funktioner (pröva och studera sista raden).

Ta bort programrader

Borttagning av en eller flera rader i ett program sker med **DEL**-kommandot (kommer av engelskans delete = ta bort, radera).

Exempel

```
DEL 30
```

tar bort programrad 30. Vårt program ser nu ut så här:

```
0010 PRINT "Djur i skogen:"
0020 PRINT "Myra"
0040 PRINT "Älg"
```

Rad 30 (den om räven) har försvunnit. För att få tillbaka programsatsen får vi inmata den på nytt.

Det spelar ingen roll att raderna 20 och 40 redan finns. Skriver du en programsats med radnummer 30 kommer datorn automatiskt att ”sortera” in den på rätt plats.

Kommandot `DEL` kan kombineras med olika slags villkor:

`DEL 50-100`

tar bort alla rader från 50 till 100.

`DEL 100-`

tar bort alla rader från 100 till programmets slut.

`DEL -200`

tar bort alla rader från programmets början till rad 200.

Vill man bli av med alla rader, dvs hela programmet, är det enklast att skriva

`NEW`

Lägg märke till skillnaden mellan ***Ctrl-Home***, som tar bort text från bildskärmen men *inte* datorns minne och kommandot `NEW`, som avlägsnar programrader från datorns minne men *inte* från bildskärmen.

Ändra i program

Vi utgår från programexemplet:

```
0010 PRINT "Djur i skogen:"
0020 PRINT "Myra"
0030 PRINT "Räv"
0040 PRINT "Älg"
```

Ett sätt att ändra en programrad är att helt enkelt skriva den på nytt. Vi kan t ex ändra rad 10 genom att skriva:

```
10 PRINT "Tre djur i skogen:"
```

Om vi nu `LIST`ar programmet ser det ut så här:

```
0010 PRINT "Tre djur i skogen:"
0020 PRINT "Myra"
0030 PRINT "Räv"
0040 PRINT "Älg"
```

Eftersom datorn bara kan ha en rad med nummer 10 i minnet, ersätter den nya raden 10 den gamla.

Ofta är det praktiskt att gå in och redigera, ändra, lägga till eller ta bort ett eller flera tecken på en rad. Med piltangenter kan man förflytta markören till rätt ställe och där göra ändringar med hjälp av tangenterna

Ins, ***Back Space*** (backsteg), ***Del*** och ***Ctrl-End***

på det sätt som redan beskrivits. Man kan också använda de mer ”primitiva sätten” att ändra och utplåna genom att skriva över med nya tecken eller trycka på mellanslag. När man gjort önskade ändringar, trycker man `↵`. Förutsatt att raden då är riktig, kommer ändringen att gälla (om den innehåller något fel rapporterar datorn detta med ett felmeddelande).

Observera följande:

- Ingen ändring registreras om raden lämnas på annat sätt än med ett tryck på ↵
- Det går bra att trycka ↵ oavsett var på raden markören står

Tillägg av nya programrader

Vi använder samma program som vi arbetat med tidigare

```
0010 PRINT "Tre djur i skogen:"
0020 PRINT "Myra"
0030 PRINT "Räv"
0040 PRINT "Älg"
```

Lås oss göra om det så att det ger bättre utskrift!

Skriv

```
5 PAGE
7 PRINT
```

och

```
15 PRINT
```

Om du LISTar programmet ska det se ut så här:

```
0005 PAGE
0007 PRINT
0010 PRINT "Tre djur i skogen:"
0015 PRINT
0020 PRINT "Myra"
0030 PRINT "Räv"
0040 PRINT "Älg"
```

Vi har tillagt rad 5, som vid körning tömmer skärmen och flyttar markören till övre vänstra hörnet. Rad 7 är en ”tom” PRINT-sats, som ger en radframmatning så att utskriften påbörjas på nästa rad.

Rad 15 är ytterligare en ”tom” PRINT-sats som ger ”luft” i utskriften.

Utskrift över hela skärmen

Bildskärmen har 25 rader och på varje rad får det plats 80 tecken (kolumner). Var och en av dessa teckenrutor kan man nå med hjälp av två *koordinater*, rad- och kolumnnummer.

Radnumret kan ligga mellan 1 och 25 och kolumnnumret mellan 1 och 80.

Teckenrutan med koordinaterna 1,1 ligger i övre vänstra hörnet. Den med koordinaterna 1,80 ligger i övre högra hörnet. Teckenrutan med koordinaterna 25,1 ligger i nedre vänstra hörnet och den med koordinaterna 25,80 i nedre högra hörnet.

För att skriva ut något med början i en viss rad och kolumn kan vi använda

```
PRINT AT <rad>,<kol>:
```

följt av det som ska skrivas ut. Skärmpositionen anges med koordinaterna, <rad> och <kol>, som ska vara heltal (det som omges av ”vinkelparenteser” ersätts med koordinaterna).

Exempel

Följande program

```
0010 PAGE
0020 PRINT AT 12,31:"Mitten"
```

skriver ordet `Mitten` mitt på skärmen. Bokstaven `M` hamnar i skärmpositionen 12,31.

För att flytta markören till en skärmposition *utan* att samtidigt skriva ut något, används satsen

```
CURSOR <rad>,<kol>
```

Första utskriften efter `CURSOR <rad>,<kol>` kommer då att börja i position `<rad>,<kol>`.

Exempel

```
0010 PAGE
0020 CURSOR 12,31
0030 PRINT "Mitten"
```

ger samma utskrift som föregående programexempel.

Omnumrera rader

Vi sa förut att det är praktiskt att ha raderna numrerade 10, 20, 30...

Det finns ett speciellt kommando för att omnumrera programrader:

```
RENUM                (av engelskans renumber = omnumrera)
```

Om du skriver

```
RENUM
```

och sedan

```
LIST
```

kan du se vad som händer. Programraderna är nu numrerade 10, 20, 30.

Det är möjligt att på olika sätt detaljstyra omnumreringen:

RENUM 100	ger radnummer som börjar på 100 med steg om 10.
RENUM ,50	ger radnummer som börjar på 50 med steg om 50.
RENUM 50,25	ger radnummer som börjar på 50 med steg om 25.
RENUM 50-;100	programmet omnumreras f o m rad 50. Första omnumrerade radnumret blir 100 med steg om 10.
RENUM 50-;100,20	programmet omnumreras f o m rad 50. Första omnumrerade radnumret blir 100 med steg om 20.

Pröva några olika alternativ och studera programmet efteråt med `LIST`. Provkör också! Som du märker händer ingenting med programutmatningen, dvs programmet fungerar som förut: Det är bara radnumreringen som ändrats.

Utskrift på skrivare

Om en skrivare är ansluten till din dator kan du få den text som finns på bildskärmen utmatad på skrivaren genom att trycka

Skift-PrtSc

På datorslang brukar detta kallas för att ”göra en skärmdump”.

Vill du ha en skrivarlistning av *hela* programmet, oavsett om det finns på skärmen eller inte, skriver du

```
LIST "prn:"PRN:
```

Det är också möjligt att ”styra om” den utmatning som åstadkoms vid programkörning till skrivare. För att erhålla skrivarutmatning lägger du in en extra programrad i ditt program:

```
10 SELECT OUTPUT "PRN:"
```

Alla PRINT-satser efter denna rad kommer att ge utskrift på skrivare i stället för på bildskärmen. Man brukar säga att ”utskriften omdirigeras till skrivaren”.

För att återföra utskriften till bildskärmen lägger du till en rad som lyder

```
30 SELECT OUTPUT "CON:"
```

Följande program

```
0010 SELECT OUTPUT "PRN:"
0020 PRINT "Hej skrivare"
0030 SELECT OUTPUT "CON:"
0040 PRINT "Hej bildskärm"
```

skriver

Hej skrivare

på skrivaren och

Hej bildskärm

på bildskärmen.

Lagra program på yttre minne

Det vore inte mycket bevänt med en dator om man var tvungen att skriva in ett program varje gång man behöver det. Därför är nästan alla datorer försedda med *yttre minnen*, vanligen skrivminnen, där program (och data) kan lagras. När ett program lagras på yttre minne får man en kopia av programmet som kan hämtas in till datorn vid behov.

Innan du lagrar ett program måste du hitta på ett lämpligt programnamn. Tyvärr kan inte namn väljas helt fritt – inskränkningarna beror inte på COMAL, utan på *operativsystemet* (DOS). Programnamnet får bestå av upp till 8 bokstäver och siffror (men inga blanktecken). Dessutom kan man ange ett s k *filtypstillägg* (eller *filtypsbeteckning*) på högst 3 tecken, som åtskiljs från de övriga tecknen med en punkt.

Låt oss döpa vårt program till PROGRAM1.

Hur vi sedan bär oss åt, beror delvis på om datorn har hårddisk eller inte. I vilket fall som helst behövs en preparerad ("formaterad") flexskiva (diskett) där du kan lagra dina program.

Med hårddisk:

Placera flexskivan i flexskivenheten (kallas A-enheten). Skriv

```
SAVE "A:PROGRAM1"
```

och tryck ↵.

Utan hårddisk:

Placera flexskivan i den lediga flexskivenheten (kallas B-enheten). Skriv

```
SAVE "B:PROGRAM1"
```

och tryck ↵.

Nu kommer skivminnesstationen med flexskivan att snurra en liten stund.

I fortsättningen gäller exemplen en dator *med* hårddisk. För det mesta är det enkelt att "översätta" till diskettfallet: Byt ut "A" mot "B" (flexskivenheten där din diskett finns) och byt ut "C" (hårddisken) mot "A" (där COMAL-systemet finns).

Om du skriver

```
DIR "A:"
```

får du en förteckning över de program (och ev övriga "filer") som finns på skivan i A-enheten. Ditt program skall nu finnas med i skivans innehållsförteckning eller DIRectory, som det heter på engelska. Lagg märke till att namnet fått ett tillägg: Det heter nu PROGRAM1.CML. Filtypsbeteckningen CML visar att det är ett COMAL-program och läggs till automatiskt av COMAL-systemet om man inte själv anger något annat tillägg.

Om du vill ta bort, radera, programmet från flexskivan skriver du

```
DELETE "A:PROGRAM1.CML"
```

Observera att fullständiga namnet måste anges, med tillägget CML.

Om du skriver

```
DIR "C:"
```

får du en lista på de program som finns i skivenhet C.

Skriver du bara

```
DIR
```

så visas vilka program som finns på den skivenhet som är *aktuell*.

När du startat en dator *på* hårddisken är alltid C-enheten aktuell. På denna finns COMAL-tolken och de program som behövs för att datorn ska starta. Vanligen har du arbetsskivan placerad i A-enheten och måste ange dess namn för att t ex spara ett program. Skrivarbetet kan dock minskas genom att skivenheten med flexskivan görs till aktuell skivenhet. Det görs med kommandot

```
UNIT "A:"
```

Då behöver du inte ange skivenheten varje gång du ska göra något med flexskivan. Skriver du nu

```
DIR
```

så erhåller du utskrift av innehållet på din flexskiva.

Hämta program från yttre minne

När du vill hämta in ett program använder du det namn som du sparade programmet med.

Under förutsättning att du gjort skivenheten med din programskiva till "aktuell enhet" (med kommandot `UNIT` enligt ovan) skriver du

```
LOAD "PROGRAM1"
```

och trycker ↵.

För att "ladda" ett program från en annan skivenhet än den aktuella, anger du även skivenhetens namn, t ex

```
LOAD "A:PROGRAM1"
```

När programmet inhämtats till datorns minne kan du kontrollera hur det ser ut med `LIST` eller köra det med `RUN`.

Vill du kontrollera vilken skivenhet som är aktuell kan du antingen skriva

```
DIR
```

varvid du erhåller en förteckning över filerna på aktuell skivenhet, eller skriva

```
PRINT UNIT
```

vilket bara ger namnet på den aktuella skivenheten.

Övning 1.15

a) Skriv in följande program och kör det.

```
10 PAGE
20 PRINT
30 PRINT "Välkommen till COMAL!"
```

b) Använd i stället satsen

```
PRINT AT
```

på programrad 30 så att texten hamnar mitt på skärmen och kör programmet.

c) Lägg till ordet 'programspråket' i texten före ordet `COMAL`.

d) Tilläggs en sats som skriver ut namnet på den skola (eller lokal), där du befinner dig.

e) Komplettera med en eller flera nya satser som skriver ut namnen på dig själv och några av dina bekanta.

Övning 1.16

a) Skriv ett program som rensar bildskärmen och därefter mitt på skärmen skriver ut texten

Fem huvudstäder i Europa

b) Komplettera med programsatser som skriver ut namnen på 5 huvudstäder. Se till att det blir "luft" i utskriften.

Övning 1.17

Skriv ett program som ger nedanstående utskrift:

En dator består av 4 huvuddelar:

- 1) Inenhet (tangentbord)
- 2) Centralenhet (processenhet och primärminne)
- 3) Yttre minne (sekundärminne)
- 4) Utenhet (bildskärm eller skrivare)

Lagra programmet på yttre minne, t ex under namnet DATORN.

Övning 1.18

Skriv ett program som ger utskriften

```

          COMAL
    är ett programspråk
    som ger möjlighet att skriva
    lättlästa och lätträttade
    program
  
```

Övning 1.19

Skriv ett program som med lämplig rubrik skriver ut en lista med förkortningar från dataområdet. Här följer några exempel:

Vanliga förkortningar inom dataområdet:

```

ADB = Automatisk DataBehandling
MDB = Manuell DataBehandling
CPU = Processenhet
ALU = Aritmetisk-logisk enhet
I/O = In/utenhet
EAN = Streckkod för prismärkning
CAD = Datorstödd konstruktion
CAI = Datorstödd inlärning
CAM = Datorstödd produktionsstyrning
PC  = Persondator
  
```

Utöka gärna listan. Lagra programmet under namnet "DATAFORK".

Övning 1.20

Skriv ett program som mitt på skärmen skriver ut

```
*****
**                               **
**  Mina memoarer                **
**                               **
*****
```

Övning 1.21

Gör ett program som skriver ut följande fråga i en frågesport.

Vad heter huvudstaden i Norge?

- a) Stockholm
- b) Oslo
- c) Åmål

Övning 1.22

Många program styrs från en *s k meny* som informerar om vad man kan göra med programmet. Skriv ett program som ger följande meny mitt på bildskärmen:

MENY

- 1. Skriv ut
- 2. Läs in
- 3. Ändra
- 0. Avsluta

Datorn räknar

Radera gammalt program från minnet (med `NEW`) och skriv programraden:

```
10 PRINT "Svar="
```

Vi har nu ett program som består av en enda programrad. Kör det! Utskriften blir:

Svar=

Detta var inte så värst mycket nytt! Ändra programsatsen till:

```
10 PRINT "Svar=" ; tal
```

Kör programmet! Utskriften blir nu:

```
Svar=
I 0010: tal: Okänd variabel
```

Det var mindre bra! Datorn ger ett felmeddelande.

Vi får anledning att införa ett nytt begrepp: **tal** är en *variabel*. Variabler används när man vill att datorn ska lagra värden.

Felmeddelandet beror på att variabeln `tal` saknar värde. Datorn vet inte vad den ska skriva ut. I COMAL måste en variabel ges ett värde första gången den används i ett program.

Ett sätt att ge en variabel ett värde är att utnyttja en *tilldelningssats*. För tilldelning används `:=` (kolon och likhetstecken). Denna teckenkombination kallas för *tilldelningsoperatör*.

De flesta BASIC-dialekter använder likhetstecknet för tilldelning, men i COMAL (liksom i flera andra moderna programspråk) är likhetstecknet reserverat för jämförelser. Även COMAL tillåter att du bara skriver ett likhetstecken vid inmatning av en tilldelningssats, men det omvandlas automatiskt till tilldelningsoperatör (`:=`) när du trycker ↵.

Skriv in raden

```
5 tal=21
```

Om du ”listar” programmet, ser det ut så här:

```
0005 tal:=21
0010 PRINT "Svar=";tal
```

OBS: *Inget* citationstecken skrivs kring variabelnamnet. Programkörning ger nu utskriften

```
Svar= 21
```

Nu fick vi inget felmeddelande. Datorn ”visste” vad den skulle göra.

I tilldelningssatsen kan man utnyttja datorns förmåga att räkna. Rad 5 kan t ex skrivas

```
0005 tal:=3*7
```

eller

```
0005 tal:=43+12+47-81
```

Tilldelningssatsen kan också utnyttjas som ett kommando, dvs datorn utför satsen direkt om den skrivs utan radnummer.

Skriv (utan radnummer):

```
tal:=3*7
PRINT "Svar=";tal
```

Datorn svarar

```
Svar= 21
```

Om namn på variabler

Ett *variabelnamn* (kallas ofta *identifierare* i datorsammanhang) består av en bokstav följt av högst 237 bokstäver, siffror eller tecknen `_` (understrykning) och `'` (apostrof).

- Namnet måste bestå av ett sammanhängande ”ord”. Vill man använda två ord måste de knyts ihop med `_` (understrykningstecken) eller `'` (apostrof), t ex `stort_tal` och `litet'tal`.
- Namnet bör väljas så att det på ett bra sätt anger vad variabeln representerar.
- Reserverade COMAL-ord får inte utnyttjas som variabelnamn.

- Val av klara och tydliga variabelnamn som är lätta att komma ihåg och begripa är en viktig del av arbetet med programkonstruktion.
- COMAL-systemet skriver variabelnamn med små bokstäver (gemener). Det spelar dock ingen roll om du skriver in dem med stora bokstäver (versaler), ty datorn gör om dem till gemener.

Här följer exempel på självförklarande variabelnamn som kan vara lämpliga i olika situationer:

```
avstånd
vinkel
decimaltal
antal_tecken
medelvärde
räntesats
```

Det är också möjligt att använda enstaka bokstäver som variabelnamn, t ex *x*, *y*, *a*, men den möjligheten bör användas sparsamt, ty det är lätt att glömma bort vad sådana namn avser.

Följande ord är *inte* tillåtna som variabelnamn, eftersom de inte börjar med en bokstav:

```
10talssiffra
0värde
(index)
```

Följande ord är *inte* tillåtna som variabelnamn, eftersom de är reserverade ord i COMAL:

```
MAXINDEX    eller  maxindex
ORD          eller  ord
PAGE        eller  page
PROC        eller  proc
```

Här följer förslag på variabelnamn som skulle kunna användas i stället för de ovanstående:

```
maxindx eller max_index
ordet
sida
procent
```

Övning 1.23

Ge förslag på variabelnamn som kan representera restid, folkmängd, tillgodohavande, antal, elevnummer och artikelnummer.

Övning 1.24

Skriv ett program som ger variabeln `antal_elever` värdet 30 och skriver ut variabelvärdet tillsammans med texten

```
Antal elever i klassen =
```

Övning 1.25

En vara kostar 3500 kr. Vid ett extraerbjudande sänktes priset med 6%. Skriv ett program som tilldelar variabeln `extrapris` värdet

```
(1-6/100)*3500
```

och skriver ut variabelvärdet tillsammans med texten

```
Extrapris
```

Övning 1.26

Skriv ett program som ger variabeln `pi_approx` värdet 355/113 och skriver ut variabelvärdet tillsammans med texten

```
Approximativt värde på pi =
```

Övning 1.27

Gör ett program som tilldelar variabeln `e` värdet $33559*9*9/1000000$ och skriver ut variabelvärdet tillsammans med texten

```
Ett bra närmevärde för talet e =
```

Olika sätt att lagra värden i datorn

De flesta programspråk kan lagra talvärden på olika sätt: Heltal på ett sätt och reella tal (tal som kan ha decimaler) på ett annat. Man säger att de tillhör olika *datatyper*. Reella tal kallas i data-sammanhang ofta *flyttal*, ett begrepp som har sin grund i det sätt på vilket de lagras i datorn.

För att i COMAL ange att man vill arbeta med ett heltal lägger man till tecknet

```
#
```

efter variabelnamnet.

Vill man använda ett flyttal skriver man ingenting efter variabelnamnet.

Exempel

Om vi i ett program skriver

```
10 dussin#:=12
20 PRINT dussin#
```

och kör det, tilldelar vi heltalsvariabeln `dussin#` värdet 12 och får det utskrivet. Prova själv.

Exempel

Skriver vi i stället

```
10 dussin:=12
20 PRINT dussin
```

och kör det kommer värdet 12 att lagras som ett flyttal. Utskriften blir dock samma som förut.

Exempel

Om vi i ett program skriver

```
10 radie:=2.8
20 PRINT radie
```

och kör det fås utskriften 2.8.

Exempel

Om vi i stället skriver

```
10 radie#:=2.8
20 PRINT radie#
```

och kör det får vi utskriften 3. Talet avrundas till ett heltal.

När vi använder heltalsvariabler märker vi ingen skillnad i början. Skillnaden kommer att märkas när en talvariabel används ofta eller när vi har stora mängder variabelvärden. Heltalsvärden hanteras snabbare och upptar mindre plats i datorns minne än flyttal. Dessutom är ett heltalsvärde alltid exakt, medan flyttalsvärden ofta bara är approximativa. Använd därför alltid heltalsvariabler när så är möjligt.

Heltalsvariabler kan dock inte alltid utnyttjas för heltalsvärden (tal utan decimaler), ty heltalsvariablernas talområde är betydligt mindre än flyttalsvariablernas.

Exempel

Om vi skriver programmet

```
10 tal#:=3000000000
20 PRINT tal#
```

och kör det får vi felutskriften

```
I 0010: Spill
```

Detta betyder att talet är för stort för att kunna lagras som ett heltal.

Heltal kan representeras i intervallet -2147483648 till 2147483647.

Flyttalsområdet är mycket större:

$$\pm 1.79769313486232 \cdot 10^{308}$$

dvs mycket stora negativa och positiva tal kan representeras.

Flyttal inom intervallet $\pm 1.0 \cdot 10^{-307}$ behandlas som noll av systemet.

Ett tal lagrat som ett flyttal har en noggrannhet av 15-16 siffror.

Stora tal (fler än 15 heltalssiffror), liksom tal som är mindre än 0.0001, skriver COMAL-systemet i exponentform. Detta innebär att det skrivs med en talfaktor, bokstaven e eller E och en exponent. Talfaktorn brukar kallas *mantissa*.

Vid inmatning kan även tal som inte är extremt stora eller små anges på exponentform, men de omvandlas då till vanlig decimalform av COMAL-systemet.

Exempel

Om vi skriver programmet

```
10 avstånd:=1.324e2
20 PRINT avstånd
```

och kör det erhålls utskriften 132.4

OBS att talvärdet direkt omvandlas till decimalform när du skrivit in programraden och tryckt ↵.

```
0010 avstånd:=132.4
0020 PRINT avstånd
```

Exempel

Skriver vi programmet

```
10 avstånd:=1.324e14
20 PRINT avstånd
```

och kör det får vi utskriften 132400000000000

Exempel

Vid körning av programmet

```
10 avstånd:=1.324e15
20 PRINT avstånd
```

fås utskriften 1.324E+15

Omvandling från exponentform till decimalform sker även för små tal, dvs då exponenten är negativ. Omvandling sker då talet är större än 0.0001.

Exempel

Med

```
0010 avstånd:=1.324e-4
```

sker omvandling till decimalform.

Med

```
0010 avstånd:=1.324e-5
```

sker ingen omvandling. Kontrollera själv detta!

OBS: Det är inte tillåtet att ha en flyttals- och en heltalsvariabel med samma namn i ett program. Körning av programmet

```
0010 tjog#:=20
0020 PRINT tjog#
0030 tjog:=20
0040 PRINT tjog
```

ger därför felmeddelandet:

```
I 0030: tjog#: Felaktig typ
```

Övning 1.28

Undersök hur olika tal skrivs av COMAL-systemet genom att fylla i nedanstående tabell.

COMAL-sats	Skärmutskrift
PRINT 6.532E4	
PRINT 6532E-3	
PRINT 12345678901234567	
PRINT 0.00005176	
PRINT 123.456*10^7	
PRINT 123.456*10^(-7)	

Gör gärna en liknande tabell och pröva andra numeriska konstanter.

Datorn stavar

Vi har hittills bara utnyttjat talvariabler, *numeriska variabler*.

Det finns också variabler vars värden är text. Sådana variabler kallas *strängvariabler*.

Här är ett program med en strängvariabel, `husdjur$`, som tilldelas värdet "Katt":

```
0010 DIM husdjur$ OF 4
0020 husdjur$:="Katt"
0030 PRINT "Smidigt djur:";husdjur$
```

Att det är fråga om en strängvariabel anges genom att variabelnamnet avslutas med `$`, dollartecknet.

Innehållet i strängvariabeln, strängen, kallas dess värde. Strängvariabler kan, liksom andra variabler, ges värden med tilldelningssatser. Strängvärden skrivs mellan citationstecken, t ex "Katt" i programrad 20.

Den plats som en *talvariabel* upptar i datorns minne har en fast storlek, oberoende av om talvariabeln lagrar ett stort eller litet tal. En *sträng* (ett textvärde) lagras däremot teckenvis, varför det krävs mer minne ju fler tecken som ingår i strängen. Därför bör en strängvariabel *dimensioneras*, vilket innebär att utrymme reserveras i datorns minne. Dimensioneringen görs på rad 10, som reserverar plats för högst fyra tecken. Strängdimensionering

Om vi inte dimensionerar en strängvariabel kommer systemet att automatiskt dimensionera den till ett standardvärde (40 tecken). Avlägsna rad 10 och provkör programmet! God programmeringssed är dock att alltid dimensionera strängvariabler. Lägg också märke till att i vårt exempel är det minnesslöseri att ej dimensionera `husdjur$` (40 tecken reserveras i stället för 4).

Exempel

Dimensionering av strängvariabeln `bokstav$` för 1 tecken och tilldelning av värdet A:

```
DIM bokstav$ OF 1
bokstav$:="A"
```

Strängvariabeln `artnamn$` dimensioneras för 25 tecken och tilldelas värdet `Mutter M 21`:

```
DIM artnamn$ OF 25
artnamn$:="Mutter M 21"
```

Ovan är strängvärdet endast 11 tecken långt och då är dimensioneringen slöseri med minnesutrymme. Vi har dock här garderat oss för att längre artikelnamn kan dyka upp.

Exempel

Strängvariabeln `space$` dimensioneras för 50 tecken och tilldelas värdet 50 blanktecken:

```
DIM space$ OF 50
space$:=" "
```

Exempel

Här dimensioneras en variabel, `stjärnrad$`, till 25 tecken och tilldelas ett värde:

```
DIM stjärnrad$ OF 25
stjärnrad$:="*****"
```

Exempel

Man kan ha både det numeriska heltalsvärdet `123` och strängvärdet `123` i samma program, naturligtvis var och en tilldelad rätt variabeltyp (OBS: Variabelnamnen måste vara olika):

```
tal#: =123
```

och

```
DIM talsträng$ OF 3
talsträng$:="123"
```

Exempel

Om `huvudstad$` dimensionerats för 6 tecken, kan den anta värdena `London`, `Paris`, `Oslo` och `Rom`. För `Stockholm` och `Lissabon` räcker det reserverade minnesutrymmet bara till för värdena `Stockh` och `Lissab`.

Övning 1.29

Föreslå variabelnamn som kan representera telefonnummer, bilmodell, veckodag, årstid, kontonummer, klassbeteckning och teckenrad.

Övning 1.30

- Dimensionera en strängvariabel om 11 tecken för personnummer och tilldela den ett värde, t ex ditt eget personnummer.
- Dimensionera en strängvariabel `svar$` så att den kan anta antingen värdet `Ja` eller `Nej`.

Övning 1.31

Dimensionera en strängvariabel för svenska alfabetet (28 tecken) och tilldela variabeln sitt värde.

Övning 1.32

Variablerna `stortdjur$`, `litetdjur$` och `listigt djur$` ska tilldelas följande värden:

```
stortdjur$ := "Älg"
litetdjur$ := "Myra"
listigt djur$ := "Räv"
```

- Skriv lämpliga dimensioneringssatser för dessa variabler
- Utöka programmet med utskriftssatser så att utskriften blir

Tre djur i skogen:

```
Älg
Myra
Räv
```

Övning 1.33

Skriv ett program som dimensionerar och ger variabeln `månad$` ett värde och variabeln `antal_dagar#` samhörande värde. Programmet ska sedan skriva ut variabelernas värden.

Övning 1.34

Skriv ett program som tilldelar variabeln `artnamn$` värdet `Mutter M 21` och variabeln `antal#` värdet `2700`. Programmet skall därefter skriva ut variabelvärdena tillsammans med texten:

```
I LAGER:
```

Övning 1.35

Komponera en matsedel med förrätt, varmrätt och efterrätt. Använd variablerna `förrätt$`, `varmrätt$` och `efterrätt$`.

Utskriften kan utformas med följande rubrik

```
Matsedel den 15 mars 1992
=====
```

Använd variabeln `datum$` för att tilldela och skriva ut dagens datum.

Övning 1.36

a) Gör ett program som skriver ut etiketter. Använd variabelnamnen `namn$`, `gadr$` och `padr$` för namn, gatuadress och postadress. Dimensionera variablerna på lämpligt sätt. Tilldela dem värden och skriv ut variabelvärdena på var sin rad.

b) Dimensionera och tilldela variabeln `perforering$` ett värde på 18 minustecken. Komplettera a) så att etiketten inramas av två rader med *minustecken*. Så här kan den se ut:

```
-----
<Namn>
<Gatuadress>
<Postadress>
-----
```

Anmärkning: Om du vill ha etiketterna utskrivna på skrivare så går det fint – se avsnittet om utskrift på skrivare ovan om du glömt hur man åstadkommer detta!

Dialog med datorn

Hittills har vi gett variabler värden med tilldelningssatser. I många sammanhang behöver man kunna ange eller ändra variabelvärden i dialog med datorn under programkörningen.

För att under pågående körning mata in ett strängvärde till variabeln `text$`, skriver du satsen:

```
0020 INPUT text$
```

När denna sats utförs stoppar programmet upp så att ett värde kan inskrivas från tangentbordet. Inmatningen avslutas med ett tryck på ↵, varefter programmet fortsätter med nästa sats.

Exempel

Vi skriver följande program:

```
10 DIM text$ OF 25
20 INPUT text$
30 PRINT "Inmatad sträng =";text$
```

Om vi kör programmet och matar in strängvärdet

```
Hokus pokus
```

```
fås utskriften
```

```
Inmatad sträng = Hokus pokus
```

Exempel

Vi skriver följande program:

```
10 INPUT antal#
20 PRINT "Dubbelt upp=";2*antal#
```

Om vi kör programmet och matar in

får vi utskriften:

```
Dubbelt upp= 40
```

Exempel

Inmatning av mer än ett *talvärde* kan ske med uppräknings

```
0010 INPUT tal1,tal2
```

Här skall två variabelvärden anges. Vid inmatning under programkörningen måste de två talvärdena skiljas åt med komma- eller blanktecken.

Ett annat och som regel bättre sätt är att inmata värdena med var sin INPUT-sats:

```
0010 INPUT tal1
0020 INPUT tal2
```

Exempel

Inmatning av värden till två eller flera strängvariabler görs alltid med var sin INPUT-sats, eftersom komma- och blanktecken kan ingå i strängar och därmed inte kan skilja strängvärdena åt.

```
0010 DIM text1$ OF 25,text2$ OF 25
0020 INPUT text1$
0030 INPUT text2$
0040 PRINT text1$
0050 PRINT text2$
```

Det är viktigt att programanvändaren vet vad som ska inmatas. Därför bör INPUT-satser alltid innehålla förklarande text (ledtext), som skrivs ut före det tillfälliga avbrottet i programmet. Ledtexten skrivs inom citationstecken efter INPUT och avslutas med ett kolon (:) följt av variabelnamnet.

Exempel

För att inmata en sträng och ett heltal kan INPUT-satserna t ex utformas så här:

```
0010 DIM text$ OF 25
0020 INPUT "Ange ett strängvärde (högst 25 tkn): ": text$
0030 INPUT "Ange ett heltal: ": heltal#
```

Lägg märke till att man i ledtexten kan ange hur många tecken som får skrivas in:

```
0010 INPUT "Ange 2 tal, åtskilda med komma-/blanktecken: ": tal1,tal2
```

Ett bättre alternativ är:

```
0010 INPUT "Ange första talet: ": tal1
0020 INPUT "Ange andra talet: ": tal2
```

Exempel Programmet

```
0010 INPUT "Ange ditt lyckotal: ": lyckotal#
0020 PRINT "Mitt lyckotal är";lyckotal#
```

ger vid körning med dialogen:

Ange ditt lyckotal: 13

utskriften

Mitt lyckotal är 13

Exempel Programmet

```
0010 DIM stortdjur$ OF 3
0020 INPUT "Ange ett stort djur (högst 3 bokstäver): ": stortdjur$
0030 PRINT stortdjur$;"är ett stort djur i skogen."
```

ger vid körning med dialogen:

Ange ett stort djur (högst 3 bokstäver): Älg

utskriften

Älg är ett stort djur i skogen.

Exempel

Följande program frågar efter datum och skriver ut detta på bildskärmens översta rad.

```
0010 DIM datum$ OF 6
0020 INPUT "Datum (ÅÅMMDD): ": datum$
0030 PAGE
0040 PRINT "Dagens datum:";datum$
```

"ÅÅMMDD" informerar om hur datum skall anges: årtal med 2 siffror, månad med 2 siffror och dag med 2 siffror.

Om programmet körs med dialogen

Datum (ÅÅMMDD): 910924

blir utskriften längst upp på "blankad skärm":

Dagens datum: 910924

Ledtexter kan placeras med början i valfri punkt på bildskärmen. Detta sker med satsen

```
INPUT AT rad#,kol#:
```

där `rad#` kan anta alla värden mellan 1 och 25 och `kol#` alla värden mellan 1 och 80, på samma sätt som `PRINT AT`-satsen.

Exempel

Om sats 20 i förra exemplet ändras till

```
0020 INPUT AT 12,25: "Datum (ÅÅMMDD): ": datum$
```

så sker dialogen

```
Datum (ÅÅMMDD): 910924
```

mitt på bildskärmen.

För att förhindra att den som kör ett program inmatar fler tecken eller siffror än avsett, kan man i INPUT-satsen ange det maximala antalet tecken som får matas in. Detta sker enligt modellen:

```
INPUT AT rad#,kol#,max#: "<ledtext>": <variabellista>
```

max# anger maximala antalet tecken som kan inskrivas när man inmatar värde(n) till den eller de variabler som ingår i variabellistan. Vid försök att skriva fler tecken vägrar markören att förflytta sig längre åt höger.

Exempel

```
0010 DIM datum$ OF 6
0020 PAGE
0030 INPUT AT 12,25,6: "Dagens datum (ÅÅMMDD): ": datum$
```

Om någon, vid körning av detta program, försöker skriva in ett datum med mer än 6 siffror blir det stopp i position 6.

Exempel

```
0010 PAGE
0020 INPUT AT 2,1,2: "Ange ditt lyckotal (1-99):": lyckotal
0030 PRINT "Mitt lyckotal är";lyckotal
```

Här kan vi endast mata in ett 2-siffrigt lyckotal.

Exempel

```
0010 PAGE
0020 INPUT AT 2,1,9: "Ange två tal, åtsk. med komma: ": tal1,tal2
```

I maxantalet tecken (här 9) inräknas det komma- eller blanktecken, som ska åtskilja de två talen. Om vi inmatar decimaltal inräknas också decimalpunkten i maxantalet tecken.

Exempel

Följande exempel visar att man kan arbeta med dialog och utskrift på valfri plats på bildskärmen:

```
0010 DIM fnamn$ OF 20, enamn$ OF 20
0020 PAGE
0030 INPUT AT 2,50,20: "Förnamn: ": fnamn$
0040 INPUT AT 3,48,20: "Efternamn: ": enamn$
0050 PRINT AT 15,1: "Efternamn",TAB(21),"Förnamn"
0060 PRINT AT 16,1: enamn$,TAB(21),fnamn$
```

Inmatningen kommer att ske i bildskärmens övre högra hörn och utskriften hamnar på skärmens vänstra del, på raderna 15 och 16. I exemplet ingår också ett nytt COMAL-ord (**TAB**). Värdet inom parentes (21 ovan) anger numret på den kolumn där utskriften ska fortsätta (TABulering). TAB kan endast utnyttjas i PRINT-satser.

Automatisk radnumrering

Radnumren har egentligen bara betydelse för att vi ska kunna infoga nya programrader mellan de gamla. Om detta kunde ordnas utan att hänvisa till radnummer skulle dessa kunna avvaras. Det finns COMAL-versioner (och andra strukturerade BASIC-versioner) där detta är genomfört.

Eftersom radnumren är av så underordnad betydelse, är det bäst om man befattar sig med dem så litet som möjligt. Med kommandot

AUTO

erhålls automatiskt radnumrering med början på 10 och med steglängden 10. Detta minskar skrivarbetet och vi kan koncentrera oss på själva programkoden.

Den automatiska radnumreringen kan avbrytas med *Ctrl-Break* (eller *Ctrl-C*). Detta blir aktuellt när vi vill infoga rader ”manuellt” inuti programmet, eller när vi är klara med inmatningen av programmet.

Om det redan finns programrader då kommandot AUTO ges, startar radnumreringen med sista radnumret + 10 (stegavståndet).

Kommandot AUTO kan specificeras på olika sätt:

AUTO 50	ger radnumreringen 50, 60, 70 osv
AUTO ,100	ger radnumreringen 10, 110, 210 osv
AUTO 50,100	ger radnumreringen 50, 150, 250 osv

Allmänt: AUTO <första radnummer> <,stegavstånd>

Om en redan befintlig rad påträffas, kommer denna rad att visas med inverterad text (mörk text på ljus botten). Trycker du då på ↵ bibehålls raden oförändrad.

Exempel

Följande program frågar efter datum, artikelnamn och antal artiklar och skriver ut en daterad ”lagerrapport”.

Innan vi börjar inskriva programmet ger vi kommandot

AUTO

Då kommer radnummer

0010

fram på bildskärmen och vi kan börja skriva in programmet:

```

0010 DIM datum$ OF 6,artnamn$ OF 20
0020 PAGE
0030 INPUT "Datum (ÅÅMMDD): ": datum$
0040 INPUT "Artikelnamn (Högst 20 tecken): ": artnamn$
0050 INPUT "Antal artiklar: ": antal#
0060 PAGE
0070 PRINT
0080 PRINT "Lagerrapport";datum$
0090 PRINT
0100 PRINT "Artikelnamn:";artnamn$
0110 PRINT "Artiklar i lager:";antal#

```

När programmets sista rad inskrivits trycker du *Ctrl-Break*.

Följande dialog

```

Datum (ÅÅMMDD): 870912
Artikelnamn (Högst 20 tecken): Mutter M 21
Antal artiklar: 2700

```

ger lagerrapporten:

```

Lagerrapport 870912

Artikelnamn: Mutter M 21
Artiklar i lager: 2700

```

Övning 1.37

Skriv ett program som frågar efter ditt namn och skriver texten

```
Välkommen till COMAL
```

omedelbart följt av ditt namn. Utskriften ska ske på bildskärmens översta rad.

Övning 1.38

Skriv ett program som frågar efter 3 vilda djurarter som finns på Skansen och deras hemland. Utskrift sker på lämplig form.

Övning 1.39

Gör ett program som börjar med att skriva ut texten

```
Recept för omelett.
```

Därefter skall man i dialog med datorn svara på frågorna

```

Antal ägg:
Mjölk (msk):
Salt (tsk):
Smör/margarin (msk):
Tillredning (högst 50 tkn):

```

Programmet skall sedan skriva ut receptet på formen

Recept för omelett.

Antal ägg: st
 Mjölk: msk
 Salt: tsk
 Smör/margarin: msk
 Tillredning:

Övning 1.40

Gör ett program som skriver ut ett brev eller meddelande med samma text till ett antal personer. Programmet ska i dialog fråga efter datum och mottagarens namn, gatuadress och postadress. Utskriften av brevet kan utformas så här:

```

                                <Datum>
<Namn>
<Gatuadress>
<Postadress>

```

```

Hej!
Vi träffas i klubbstugan på torsdag
kl 18.00 för taktikgenomgång inför matchen.
Laban

```

Datorn räknar mera

Följande program demonstrerar de enkla räkneoperationerna. Det frågar efter två tal och beräknar och skriver ut värdet på talens summa, differens, produkt, kvot och potens.

```

0010 INPUT "Ange ett heltal: ": tal1#
0020 INPUT "Ange ett heltal till: ": tal2#
0030 PRINT "Summan =";tal1#+tal2#
0040 PRINT "Differensen =";tal1#-tal2#
0050 PRINT "Produkten =";tal1#*tal2#
0060 PRINT "Kvoten =";tal1#/tal2#
0070 PRINT "Potensen =";tal1#^tal2#

```

Om programmet körs med dialogen

```

Ange ett heltal: 3
Ange ett heltal till: 5

```

erhålls utskriften

```

Summan = 8
Differensen = -2
Produkten = 15
Kvoten = 0.6
Potensen = 243

```

Exempel

Tilldelningssatsen kan även utnyttjas som ett kommando. Med PRINT-kommandot kan vi sedan skriva ut värden på uttryck där variabler ingår:

Skriv (utan radnummer)

```
radie:=5.0
PRINT "Areal=";3.14*radie*radie
```

Datorn svarar

```
Areal= 78.5
```

Exempel

Vi har redan talat om den ordning i vilken räkneoperationer utförs när det förekommer flera i ett uttryck. Följande program illustrerar några av dessa regler.

```
0010 PRINT "Mata in 3 tal!"
0020 INPUT "Första talet: ": tal1
0030 INPUT "Andra talet: ": tal2
0040 INPUT "Och tredje: ": tal3
0050 svar1:=tal1/tal2*tal3
0060 svar2:=tal1/(tal2*tal3)
0070 svar3:=tal1+tal2^tal3
0080 svar4:=(tal1+tal2)^tal3
0090 PRINT "Svar1 =";svar1
0100 PRINT "Svar2 =";svar2
0110 PRINT "Svar3 =";svar3
0120 PRINT "Svar4 =";svar4
```

Kör vi programmet med dialogen

```
Mata in 3 tal!
Första talet: 9e2
Andra talet: 1.6e3
Och tredje: 5e-1
```

får vi utskriften

```
Svar1 = 0.28125
Svar2 = 1.125
Svar3 = 940
Svar4 = 50
```

Om du jämför räkneuttrycken i programmet med körningsresultatet får du en bra förklaring på i vilken ordning räkneoperationerna utförs.

En räkneoperation får aldrig vara underförstådd. I matematiken skriver man t ex ofta $2x$ i stället för $2 \cdot x$, liksom $3(x-5)$ i stället för $3 \cdot (x-5)$. Vid programmering måste multiplikationstecknen skrivas ut, dvs $2 * x$ resp $3 * (x - 5)$.

Glöm heller inte prioritetsreglerna, vilka kan kräva extra parenteser, t ex när man ska dividera 2 uttryck, där täljaren och/eller nämnaren består av flera termer.

Mera om tilldelningssatsen

Man kan se det som att värdet till höger om `:=` (tilldelningsoperatör) kopieras till variabeln till vänster:

```
antal#:=45
```

Värdet 45 på höger sida ”kopieras” till variabeln på vänster sida.

Som vi redan sett, kan det som står till höger om tilldelningsoperatör vara ett beräkningsuttryck.

```
antal#:=5*5-1
```

Uttrycket i högerledet beräknas och sedan överförs resultatet till variabeln i vänsterledet.

Man kan även ha variabler i högerledet (under förutsättning att de redan getts värden):

```
antal#:=x#+5
```

Värdet av uttrycket i högerledet måste dock ha samma *datatyp* som variabeln till vänster (undantag: hel- och flyttal kan ”blandas”). Man kan t ex *inte* tilldela heltalsvariabeln `antal#` värdet `svar$`, ty `svar$` är en strängvariabel.

Eftersom en variabel motsvarar ett visst ”fack” i datorns minne, kan vi utföra beräkningar på *variabeln själv* och därefter, med tilldelningsoperatör, lagra det nya värdet på samma plats som där det gamla låg. Detta är en intressant egenskap hos variabler som ofta kommer till användning i programmering.

Här följer exempel på några i praktiken vanligt förekommande beräkningsuttryck av ovan nämnda slag. Vi ger det först i ”vardagsspråk” och därefter med tilldelningsoperatör, som COMAL förstår.

Vardagsspråk	COMAL-sats
Öka ett värde med 1. Uttrycket brukar kallas för ”varvräknare”	<code>antal#:=antal#+1</code>
Addera ett värde	<code>sum:=sum+tal</code>
”Lägga på moms”	<code>belopp:=belopp+belopp*moms/100</code>
”Ta ut pengar”	<code>saldo:=saldo-trans</code>
Fördubbla ett värde	<code>d#:=2*d#</code>
Tredubbla ett värde	<code>x#:=3*x#</code>
Halvera ett värde	<code>b:=b/2</code> eller <code>b:=0.5*b</code>
Byta tecken på ett värde	<code>tall:=-tall</code>

Exempel

```
0010 INPUT "Ange variabelvärde: ": antal#
0020 antal#:=antal#+1
0030 PRINT "Nytt variabelvärde =";antal#
```

Körs programmet med dialogen

Ange variabelvärde: 100

erhålls utskriften

Nytt variabelvärde = 101

Användarvänlig utskrift

Vid utskrift med `PRINT`-satsen har vi begränsade möjligheter att få snygga och lättlästa utskriften.

För att taltabeller ska bli lättlästa vill vi ofta ha dem utskrivna med jämn högerkant.

En tabell med heltal vill vi inte få utskrivna så här:

```
1
8
27
64
125
```

utan så här:

```
  1
  8
 27
 64
125
```

Decimaltal, t ex belopp i en faktura, vill vi inte få utskrivna så här:

```
335.25
63.5
1350.6
```

utan så här:

```
 335.25
  63.50
1350.60
```

Med `PRINT USING` kan vi styra utseende eller format på utskriften. Det önskade formatet specificeras med en *formatsträng*, där `#`-tecknet används för att reservera plats för de sifferpositioner man anser sig behöva. Efter formatsträngen skrivs ett kolon (`:`) och därefter talvariabeln.

Tecknet `#` kallas ibland ”brädstapel”. I engelskspråkig litteratur används det som nummerbeteckning.

Exempel

```
PRINT USING "###": tal#
```

ger utskrift av högerjusterade, 1-3-siffriga heltal. Om värdet är 1- eller 2-siffrigt, så erhålls blanktecken i de första sifferpositionerna.

```
0010 INPUT "Ange ett heltal, högst 3-siffrigt: ": tal1#
0020 INPUT "Ange ett heltal, högst 3-siffrigt: ": tal2#
0030 INPUT "Ange ett heltal, högst 3-siffrigt: ": tal3#
0040 PRINT USING "###": tal1#
0050 PRINT USING "###": tal2#
0060 PRINT USING "###": tal3#
```

Kör vi programmet och matar in 2, 32 och 128 får vi utskriften

```
  2
 32
128
```

Om vi i stället matar in -8, -888 och 1000 blir utskriften

```
  -8
-888
***
```

Asteriskerna anger att det inte fanns plats för att skriva ut värdet. För att klara 1000 behövs fyra #-tecken.

Observera att -888 utskrivs, trots att det går åt 4 teckenpositioner, dvs minustecknet ”räknas inte” (bara siffrorna). För att även reservera plats för minustecknet skriver man "-###". Pröva!

Vill vi ha utskrift av decimaler anger vi decimaltecknets plats genom att placera en punkt där vi vill ha det. Satsen

```
40 PRINT USING "#####.##": tal1
```

reserverar plats för upp till 5 heltalssiffror och 2 decimaler i det värde som tal1 antar.

Skriv in följande program:

```
0010 INPUT "Ange ett tal: ": tal1
0020 INPUT "Ange ett tal: ": tal2
0030 INPUT "Ange ett tal: ": tal3
0040 PRINT USING "#####.##": tal1
0050 PRINT USING "#####.##": tal2
0060 PRINT USING "#####.##": tal3
```

Kör vi programmet med dialogen

```
Ange ett tal: 2.339
Ange ett tal: 500
Ange ett tal: 525344
```

blir utskriften

```
  2.34
 500.00
*****
```

Decimaltal avrundas vid utskrift med `PRINT USING`. Om decimaler saknas skrivs nollor ut. Skulle värdet ej rymmas i det angivna formatet anges detta med asterisker (*).

Nedan ges ett par lite mer ”finessrika” exempel på hur `PRINT USING`-satsen kan utnyttjas.

Exempel

Text kan bifogas i ”formatsträngen”, ty alla tecken utom de som gäller själva talformatet (#. -) skrivs ut oförändrade. Dessutom kan flera tal skrivas ut med samma `PRINT USING`-sats.

Följande program illustrerar båda sakerna:

```
0010 INPUT "Ange pris med moms: ": pris_med_moms
0020 moms:=0.2*pris_med_moms
0030 pris:=pris_med_moms-moms
0040 PRINT USING "Pris utan moms ###.## kr, moms ##.## kr": pris,moms
```

Lägg märke till att talvariablerna skiljs åt av kommatecken.

Exempel

`USING` kan även kombineras med `PRINT AT`

```
PRINT AT 5,10: USING "####.###": 180/PI
```

Observera placeringen av `USING`, samt de två kolontecknen. Den matematiska konstanten π skrivs `PI` i `COMAL`.

Vid arbete med text får man utnyttja andra metoder för att få lättlästa utskrifter. Följande program innehåller några nyheter.

```
0010 DIM fnamn$ OF 20,enamn$ OF 20
0020 INPUT "Förnamn: ": fnamn$
0030 INPUT "Efternamn: ": enamn$
0040 PRINT
0050 ZONE 20
0060 PRINT "Efternamn";"Förnamn"
0070 PRINT enamn$;fnamn$
```

På rad 10 dimensioneras två textvariabler, `fnamn$` och `enamn$`, för 20 tecken vardera. Satsen `ZONE 20`, på rad 50, medför att utskrifter därefter tabuleras med 20 teckenpositioner. Mellan de värden som ska skrivas med tabulering sätter man semikolon (;).

Om du kör programmet och inmatar förnamnet `Jan` och efternamnet `Larsson` blir utskriften:

```
Efternamn      Förnamn
Larsson        Jan
```

`ZONE 20` innebär att nästa utskrift kommer i position 21.

Ett värde på `ZONE` gäller tills dess man ändrar det eller skriver `NEW`. Efter `NEW` och när du startar `COMAL` är `ZONE 1` och då ger semikolon ett blanktecken mellan värdena. Om inget mellanrum önskas används kommatecken (,) i stället för semikolon.

I PRINT-satser kan vi även använda **TAB**(kol#). Värdet på kol# anger hur många steg utskriften sker från vänstermarginalen, TABulering. Här följer ett program som illustrerar detta:

```
0010 DIM fnamn$ OF 20, enamn$ OF 20
0020 INPUT "Förnamn: ": fnamn$
0030 INPUT "Efternamn: ": enamn$
0040 PRINT
0050 ZONE 20
0060 PRINT "Efternamn";"Förnamn"
0070 PRINT enamn$,TAB(20),fnamn$
```

Observera att vi på rad 70 använder kommatecken i stället för semikolon. Detta för att vi där använder TAB och inte önskar utnyttja ZONE. Vid körning av programmet och inmatning av förnamnet Jan och efternamnet Larsson blir utskriften:

```
Efternamn      Förnamn
Larsson        Jan
```

Om du i PRINT-satsen på rad 70 ändrar till TAB(21) erhålls utskriften:

```
Efternamn      Förnamn
Larsson        Jan
```

Här följer ett program som använder både PRINT USING och ZONE för att skriva ut en enkel ”lagervärdering”:

```
0010 DIM datum$ OF 6, artnamn$ OF 15
0020 PAGE
0030 INPUT "Datum (ÅÅMMDD): ": datum$
0040 INPUT "Artikelnamn (högst 15 tecken): ": artnamn$
0050 INPUT "Pris: ": pris
0060 INPUT "Antal artiklar: ": antal#
0070 PAGE
0080 PRINT
0090 ZONE 16
0100 PRINT "Lagervärdering";datum$
0110 PRINT
0120 PRINT "Artikelnamn";"Antal  Pris/st      Värde"
0130 PRINT artnamn$;
0140 PRINT USING "#####  #####.##  #####.##": antal#,pris,antal#*pris
```

Kör vi programmet med dialogen:

```
Datum (ÅÅMMDD): 870925
Artikelnamn (högst 15 tecken): Flexskiva
Pris: 11.25
Antal artiklar: 1105
```

blir utskriften:

```
Lagervärdering 70925
```

```
Artikelnamn      Antal  Pris/st      Värde
Flexskiva        1105    11.25    12431.25
```

Övning 1.41

Vilket värde har respektive variabel efter nedanstående satser

- `antal#:=antal#+1` om variabelns aktuella värde är 5
- `x#:=x#-100` om variabelns aktuella värde är 101
- `rfaktor:=rfaktor-1/100` om variabelns aktuella värde är 11.25
- `intlängd:=intlängd/10` om variabelns aktuella värde är 0

Övning 1.42

Gör ett program som frågar efter 3 tal och beräknar och skriver ut

- deras summa
- deras medelvärde
- summan av deras kvadrater
- summan av deras kuber

Övning 1.43

Gör ett program som frågar efter 1 tal och beräknar och skriver ut

- dess halverade värde
- dess värde multiplicerat med 3 och adderat med 1
- dess värde ökat med 50%
- hundra delen av dess värde

Övning 1.44

- Skriv ett program som beräknar triangelarean enligt formeln $\text{arean} = (\text{basen} \cdot \text{höjden}) / 2$. Det ska fråga efter basen och höjden och skriva ut resultatet:

Arean =

- Skriv ett program som beräknar klotvolym. Programmet ska fråga efter klotets radie och sedan beräkna och skriva ut klotets volym enligt formeln $\text{volym} = 4 \cdot \pi / 3 \cdot \text{radie}^3$.

Övning 1.45

Skriv ett program som omvandlar Celcius-grader till Fahrenheit-grader. Det ska fråga efter

`tempc`

och beräkna och skriva ut

`tempf`

enligt formeln $\text{tempf} = 9 \cdot \text{tempc} / 5 + 32$

Övning 1.46

Gör ett program som frågar efter belopp och moms (eller har moms tilldelat i programmet) och beräknar och skriver ut

Belopp

Belopp inkl. moms

Övning 1.47

Gör ett program som frågar efter artikelnummer, artikelns styckpris och antalet artiklar. Därefter skall en enkel faktura skrivas ut med rubrikraden

```
Artikelnummer   Antal   Styckpris   Totalbelopp
```

Använd PRINT USING-satsen!

Övning 1.48

Gör ett program som efterliknar en liten bank med en kund. Låt kundens tillgodohavande, saldo, vara 5000 kr. Fråga efter uttag, transaktion, och beräkna och skriv ut det nya tillgodohavandet.

Övning 1.49

Med följande inmatningssats

```
INPUT "Ange uttag/insättning med - eller + framför beloppet: ":trans
```

och satsen

```
saldo:=saldo+trans
```

kan man med programmet i övning 1.48 göra både uttag och insättning. Fullborda programmet.

Stränghantering

Hantering av strängar är en viktig aspekt i text- och ordbehandling. Vi ska nu undersöka några strängoperationer som finns i COMAL.

Strängsammanfogning

Två eller flera delsträngar kan sammanslås ("konkateneras") till en enda med operatoren +. Delsträngarna kan också vara enstaka tecken.

Nedanstående program frågar efter för- och efternamn och skriver ut hela namnet, med efternamnet först och ett kommatecken mellan efter- och förnamnet:

```
0010 DIM fnamn$ OF 10,enamn$ OF 10,namn$ OF 22
0020 INPUT "Förnamn (högst 10 tkn): ": fnamn$
0030 INPUT "Efternamn (högst 10 tkn): ": enamn$
0040 namn$:=enamn$+", "+fnamn$
0050 PRINT namn$
```

På programrad 40 infogar vi ", " mellan enamn\$ och fnamn\$. Detta för att få kommatecken och mellanslag mellan efter- och förnamnet. Om vi kör programmet med dialogen:

```
Förnamn (högst 10 tkn): Niklaus
Efternamn (högst 10 tkn): Wirth
```

fås utskriften

```
Wirth, Niklaus
```

Delsträngar

Datorn kan även plocka ut delar av strängar.

För att demonstrera detta utgår vi från nedanstående sträng:

```
text$:= "Datalogi"
      12345678 <- positioner i strängen
```

```
0010 DIM text$ OF 8
0020 text$:="Datalogi"
0030 PRINT text$(1:2)
```

ger utskriften

Da

Den allmänna regeln för hur delsträngar bildas kan vi uttrycka så här:

Delsträngen `strängvar$(startpos#:slutpos#)` får värdet från tecknet i `startpos#` till tecknet i `slutpos#` av de tecken som ingår i värdet på `strängvar$`. För att få rimliga värden krävs att `startpos#>=1` och `slutpos#>=startpos#`.

Här följer fler exempel för att klargöra det hela:

```
delsträngen text$(1:4) har värdet Data
delsträngen text$(1:1) har värdet D
delsträngen text$(4:4) har värdet a
delsträngen text$(5:) har värdet logi
```

Det sista exemplet visar att slutpositionen kan utelämnas (då medtas resten av strängen).

En delsträng kan hanteras på samma sätt som en vanlig sträng. Man kan tilldelas den till en annan strängvariabel:

```
0010 DIM efternamn$ OF 10, namn$ OF 20
0020 namn$:="Kalle Anka"
0030 efternamn$:=namn$(7:)
0040 PRINT efternamn$
```

Vilken utmatning erhålls vid körning av programmet?

Man kan också byta ut ett eller flera tecken inuti en sträng, enligt modellen:

```
0010 DIM namn$ OF 10
0020 namn$:="Kalle Anka"
0030 PRINT "Gammalt namn:";namn$
0040 namn$(3:5):="jsa"
0050 PRINT "Nytt namn:";namn$
```

På rad 40 utbyts bokstäverna i positionerna 3-5 i strängen `namn$`. Vad blir resultatet? Provkör!

Talet 4711 och *textsträngen* "4711" är helt olika saker. Med *talet* 4711 kan vi räkna (addera, dividera osv), men t ex inte sammanfoga det med en strängvariabel eller plocka ut "delsträngen" 47. Strängen "4711" kan däremot manipuleras som alla andra strängar, men vi kan inte räkna med den. Förmågan att skilja mellan och korrekt hantera olika typer av variabler/data tillhör en programmerares grundläggande färdigheter.

Övning 1.50

Antag att `filnamn$` har värdet `FIL1`. Vad blir värdet på `filnamn$` efter satsen

- a) `filnamn$:=filnamn$+".DAT"`
- b) `filnamn$:="C:"+filnamn$`

Övning 1.51

Om

```
text1$:="COMmon"
text2$:="Algorithmic"
text3$:="Language"
```

Vad blir då värdet på `text$`?

```
text$:=text1$(1:3)+text2$(1:1)+text3$(1:1)?
```

Övning 1.52

Gör ett program som frågar efter ett ord på 3 bokstäver och skriver ut ordet med alla bokstäverna dubblerade.

Övning 1.53

Skriv ett program som ur texten (strängen)

```
American Standard Code for Information Interchange
```

plockar ut förkortningen `ASCII` och skriver ut denna.

Övning 1.54

Skriv ett program som frågar efter efternamn och förnamn (tilltalsnamnet) och skriver ut:

- a) efternamnet följt av initialen i tilltalsnamnet
- b) förnamnet följt av första bokstaven i efternamnet och en punkt
- c) förkortning (signatur) bestående av första bokstaven i förnamnet och de två första bokstäverna i efternamnet.

Övning 1.55

Vad skrivs ut då nedanstående program körs. Försök besvara frågorna *innan* du provkör dem!

- a)


```
0010 DIM namn$ OF 15
0020 namn$:="Kalle"
0030 PRINT "Namn:";
0040 PRINT namn$
```

b)

```
0010 DIM adress$ OF 10
0020 adress$:="Skolkgatan 15"
0030 PRINT adress$
```

c)

```
0010 DIM sträng$ OF 8
0020 sträng$:="storkar"
0030 PRINT sträng$;"är";sträng$(1:4)+"a"
```

Övning 1.56

Följande programavsnitt finns i ett COMAL-program:

```
0100 nummer$:="1234567890"
0110 ett$:=nummer$(1:2)
0120 två$:=nummer$(5:6)
0130 tre$:=ett$+två$
0140 PRINT ett$;"+";två$;"=";tre$
```

Vilken utmatning ger programavsnittet? Försök lösa uppgiften utan att använda datorn. Kontrollera sedan ditt svar genom att skriva in och köra programmet.

Övning 1.57

Följande programrader är givna:

```
0010 DIM text$ OF 30
0020 text$:="Han skrev romanen"
0030 PRINT text$
```

Tilllägg programsatser som ändrar strängens värde till

Hon skrev romanen åt honom

och skriver ut den nya strängen.

För att göra det hela lite intressantare (och lärorikare) är det förbjudet att göra tilldelningen `text$:="Hon skrev romanen åt honom"`, även om det här skulle vara enklast. Utgå i stället från den befintliga strängen och gör nödvändiga manipulationer.

Övning 1.58

Arean av en parallelltrapets beräknas med formeln $A=h\cdot(a+b)/2$, där A =arean, h =höjden och a resp b är de parallella sidorna. Skriv ett program som beräknar och skriver ut resultatet snyggt (använd `PRINT USING`). Värdena på h , a och b ska inmatas från tangentbordet.

Övning 1.59

Skriv ett program som omvandlar m/s till km/h. Programmet ska fråga efter hastigheten i m/s och skriva ut resultatet i formen:

3.7 m/s = 13.32 km/h

NOTERINGAR

COMal ihåg**EDITERING m m:**

<i>BackSpace</i>	Raderar ett tecken till vänster om markören
<i>Ctrl-Break</i>	Avbryter pågående arbete
<i>Ctrl-End</i>	Raderar alla tecken till höger om markören
<i>Ctrl-Home</i>	Rensar bildskärmen
<i>Del</i>	Raderar ett tecken där markören står
<i>End</i>	Flyttar markören till radslut
<i>Home</i>	Flyttar markören till radbörjan
<i>Ins</i>	Växlar mellan infognings- och överskrivningsläge
<i>Skift-PrtSc</i>	Ger utskrift av skärminnehåll på skrivare

VARIABLER:

<code>:=</code>	Tilldelning
<code>feber:=38.4</code>	Variabelvärdet 38.4 representeras som ett flyttal
<code>tal#:=99</code>	Variabelvärdet 99 representeras som ett heltal
<code>text\$:="Hej"</code>	Strängvariabeln <code>text\$</code> ges värdet Hej

PROGRAMMERING (kommandon och programsatser):

AUTO	Ger automatisk radnumrering
BYE	Lämna COMAL-systemet
DEL radnr#	Tar bort programsats med angivet radnummer
DELETE "filnamn.typ"	Raderar filen FILNAMN.TYP från skivan i aktuell skivenhet
DIM...OF...	Anger maximalt antal tecken i en strängvariabel
DIR	Listar alla filnamn på aktuell skivenhet
DIR "A:"	Listar alla filnamn på skivenhet A:
END	Slut på programmet (behövs vanligen ej)
INPUT	Inmatning till datorn
INPUT AT rad#,kol#,max#:	Inmatning till datorn från position rad#, kol# med max# antal tecken
LIST	Ger programlistning på bildskärm
LIST "PRN:"	Ger programlistning på skrivare
LOAD "prognamn"	Hämtar programmet PROGNAME.CML till minnet (från aktuell skivenhet)
NEW	Raderar programmet från arbetsminnet
PAGE	Rensar skärmen och placerar markören i position 1,1
PRINT	Skriver på bildskärm eller skrivare
PRINT AT rad#,kol#:	Skriver på bildskärmen med början i position rad#,kol# ($1 \leq \text{rad\#} \leq 25$ och $1 \leq \text{kol\#} \leq 80$)
PRINT USING	Skriver ut värde hos talvariabel med angivet format
RENUM	Omnummererar programrader
RUN	Kör programmet i minnet
SAVE "prognamn"	Sparar program på aktuell skivenhet under namnet PROGNAME.CML
SELECT OUTPUT "CON:"	Programutskrift sker på bildskärm.
SELECT OUTPUT "PRN:"	Programutskrift sker på skrivare.
SIZE	Informerar om hur mycket minne som är upptaget av program och data och hur mycket minne som är ledigt.
TAB(kol#)	Flyttar markören framåt till kolumn kol#. Används enbart i PRINT-satser.
UNIT "A:"	Anger för datorn att skivenheten A: är aktuell enhet
ZONE num#	Ger angiven indelning (num#) i sidled

NOTERINGAR

Kapitel 2

Inledning

Huvuddelen av de följande sidorna ägnas åt UniCOMAL-versioner av lärobokens grafikprogram (sidorna 85-110 i Olssons bok), men inledningsvis ges också några allmänna råd och tips för att underlätta användning av bokkapitlet.

1) Starta grafikenGrafik

I Compis-COMAL, åtminstone den version som beskrivs i Olssons bok, måste ”grafikpaketet” kopplas till COMAL med

```
LINK "GRAFIK"
```

Grafiken måste därefter startas (initieras) på det sätt som beskrivs på sidorna 64-65 (om man arbetar i ”direktmod”) och 84-85 (om man skriver ett program innehållande grafik). I UniCOMAL är det enklare. I båda fallen behövs enbart

```
USE graphics
```

vilket således även startar grafiken.

2) Grafikskärmen

Ett problem är att de bildskärmar som förekommer med PC-datorer sällan har samma ”upplösning” som Compis-skärmarna (640 punkter i x-led och 400 punkter i y-led – se läroboken sid 61). Det enklaste sättet att komma runt problemet är att ”ställa om grafiksystemet” så att det arbetar med samma koordinatsystem som på Compis. Detta görs med instruktionen

```
window(0,639,0,399)
```

Detta ”trick” utnyttjas flitigt i programexemplen nedan.

3) Skillnader i grafikinstruktioner

De flesta av de grafikinstruktioner som redovisas i Olssons bok heter likadant i UniCOMAL, men det finns också väsentliga skiljaktigheter. Exempel:

- De olika varianterna av **penmode** som beskrivs på sidorna 67-68 finns inte i UniCOMAL. Däremot kan man åstadkomma liknande resultat med **pencolor**-instruktionen (se manual om du är intresserad).
- **charsize**, **chardir** och **chartype** saknas i UniCOMAL. En del av det som är möjligt att utföra med dessa instruktioner kan man åstadkomma med UniCOMAL-instruktionerna **textstyle** och **setfont** (dock t ex inte vrida texten 45 grader, utan bara 90 grader).
- **rectangle**, **rectfill** och **circfill** (sid 74) saknas. Man kan dock relativt enkelt själv skriva rutiner i COMAL som åstadkommer detta (se programexemplen nedan).

Det finns även en del instruktioner som har samma effekt men heter olika. Exempel på detta finner du om du jämför programexemplen nedan med de i boken.

Programexemplen

Sid 85:

```
0005 USE graphics
0030 plot(320,100)
```

Sid 86:

```
0005 USE graphics
0006 window(0,639,0,399) // Ger samma koordsystem som Compis
0010 //
0020 // clearscreen - Onödigt här
0030 plot(320,200)
0040 PRINT AT 14,41: "P"
```

Sid 87:

```
0010 INPUT "Ange punktens koordinater (0<=x<=639,0<=y<=399): ": x#,y#
0020 USE graphics
0030 window(0,639,0,399) // Ger samma koordsystem som Compis
0040 plot(x#,y#)
0050 // Först text under punkten.
0060 text_y#:=y#-21
0070 text_x#:=x#-4
0080 plotttext(text_x#,text_y#,"P")
0090 // Sedan text ovanför.
0100 text_y#:=y#+5
0110 plotttext(text_x#,text_y#,"P")
0120 // Text till vänster om punkten.
0130 text_y#:=y#-(16/2) // Centrerung i höjddled. Jmf text_x#:=x#-(8/2)
0140 text_x#:=x#-13 // Dvs text_x#:=x#-8-5. Jmf text_y#:=y#-13
0150 plotttext(text_x#,text_y#,"P")
0160 // Och slutligen text till höger om punkten.
0170 text_x#:=x#+5
0180 plotttext(text_x#,text_y#,"P")
```

Sid 88:

```
0010 USE graphics
0020 window(0,639,0,399) // Ger samma koordsystem som Compis
0030 INPUT "Ange bokst placering (1<=rad<=25,1<=kol<=80):": rad#,kol#
0040 PRINT AT rad#,kol#: "P"
0050 y#:=400-16*rad#
0060 x#:=(kol#-1)*8
0070 plot(x#,y#) // Teckenrutans nedre vänstra hörn
0080 plot(x#,y#+15) // Övre vänstra hörnet
0090 plot(x#+7,y#+15) // Övre högra hörnet
0100 plot(x#+7,y#) // Och slutligen teckenrutans nedre högra hörn
```

Sid 88:

```

0010 USE graphics
0020 window(0,639,0,399)
0030 FOR i#:=1 TO 10 DO
0040   INPUT AT 1,1: "Ange pktens koord (0<=x<=639,0<=y<=399):": x#,y#
0050   plot(x#,y#)
0060 ENDFOR i#

```

Sid 90:

```

0010 USE graphics
0020 window(0,639,0,399)
0030 FOR x#:=300 TO 324 DO
0040   FOR y#:=300 TO 324 DO
0050     plot(x#,y#)
0060   ENDFOR y#
0070 ENDFOR x#
0080 text_x#:=(300+(324-300)/2)-(8*4/2)+1
0090 // Jmf rad 80 med text_x#:(mittpunkt#)-(textlängd#/2)+1
0100 text_y#:=300-21 // Jmf y#:=y#-21
0110 plottext(text_x#,text_y#,"RUTA")

```

Sid 91:

```

0010 USE graphics
0020 window(0,639,0,399)
0030 FOR i#:=1 TO 250 DO
0040   x#:=RND(0,639)
0050   y#:=RND(0,399)
0060   plot(x#,y#)
0070 ENDFOR i#

```

Sid 92 (1:a ex):

```

0010 USE graphics
0020 window(0,639,0,399) // Ger samma koordsystem som Compis
0030 FOR i#:=0 TO 180 DO
0040   left(2)
0050   pendown
0060   forward(RND(150,200))
0070   penup
0080   setxy(320,200)
0090 ENDFOR i#

```

Sid 92 (2:a ex):

```

0005 // USE graphics
1000 PROC nyttritblock
1020   clearscreen
1030 ENDPROC nyttritblock

```

Sid 95 (1:a ex):

```
1000 PROC rektangel(x#,y#,l#,h#)
1010   rectangle(x#,y#,x#+l#,y#+h#)
1020 ENDPROC rektangel
```

Sid 95 (2:a ex):

```
0005 USE graphics
0010 // Huvudprogram
0015 window(0,639,0,399) // Compis-skärm
0020 // nyttritblock - Behöver vi inte anropa här
0030 FOR i#:=1 TO 100 DO
0040   rektangel(RND(50,550),RND(50,300),RND(5,50),RND(5,50))
0050 ENDFOR i#
0060 plotttext(320-(8*14/2),16,"Konstnär okänd")
0070
0080 // Deklarationsdel
0090 // Procedurer
0100
0110 PROC nyttritblock
0130   clearscreen
0140 ENDPROC nyttritblock
0150
0160 PROC rektangel(x#,y#,l#,h#)
0170   rectangle(x#,y#,x#+l#,y#+h#)
0180 ENDPROC rektangel
0190
0199 // Följande procedur är ej inbyggd i UniCOMAL
0200 PROC rectangle(x1#,y1#,x2#,y2#)
0210   moveto(x1#,y1#)
0220   drawto(x2#,y1#)
0230   drawto(x2#,y2#)
0240   drawto(x1#,y2#)
0250   drawto(x1#,y1#)
0260 ENDPROC rectangle
```

Sid 96:

```

0010 // Huvudprogram
0010 USE graphics // nyttritblock - Behövs inte här
0030 INPUT "Ange startpunkt (nedre vänstra hörnet), x,y: ": x#,y#
0040 INPUT "Ange längd: ": l#
0050 INPUT "Ange höjd: ": h#
0060 FOR i#:=1 TO 10 DO
0070     rektangel(x#,y#,l#,h#)
0080     x#:=x#+1
0090     y#:=y#+2
0100 ENDFOR i#
0110
0120 // Deklarationsdel
0130 // Procedurer
0190
0200 PROC rektangel(x#,y#,l#,h#)
0210     rectangle(x#,y#,x#+l#,y#+h#)
0220 ENDPROC rektangel
0230
0239 // Saknas inbyggd i UniCOMAL
0240 PROC rectangle(x1#,y1#,x2#,y2#)
0250     moveto(x1#,y1#)
0260     drawto(x2#,y1#)
0270     drawto(x2#,y2#)
0280     drawto(x1#,y2#)
0290     drawto(x1#,y1#)
0300 ENDPROC rectangle

```

Sid 97:

```

1000 PROC likbent(x#,y#,b#,h#)
1010     moveto(x#,y#)
1020     draw(b#,0)
1030     draw(-b#/2,h#)
1040     draw(-b#/2,-h#)
1050 ENDPROC likbent

```

Sid 98 (1:a ex):

```

2000 PROC liksidig(x#,y#,s#,v#)
2010     moveto(x#,y#)
2020     setheading(v#)
2030     forward(s#)
2040     left(120)
2050     forward(s#)
2060     left(120)
2070     forward(s#)
2080 ENDPROC liksidig

```

Sid 98 (2:a ex):

```
0010 // Huvudprogram
0015 USE graphics
0016 window(0,639,0,399) // Ger samma koordsystem som Compis
0017
0020 // nyttritblock - behövs inte här
0030 FOR i#:=0 TO 5 DO
0040     liksidig(300,200,100,60*i#)
0050 ENDFOR i#
0060
0970 // Deklarationsdel
0980 // Procedurer
0990
1000 PROC nyttritblock
1020     clearscreen
1030 ENDPROC nyttritblock
1990
2000 PROC liksidig(x#,y#,s#,v#)
2010     moveto(x#,y#)
2020     setheading(v#)
2030     forward(s#)
2040     left(120)
2050     forward(s#)
2060     left(120)
2070     forward(s#)
2080 ENDPROC liksidig
```

Sid 100:

```
0005 USE graphics
0006 window(0,639,0,399) // Ger samma koordsystem som Compis
0010 // Huvudprogram
0020 // nyttritblock Behövs ej här
0030 stol
0040
0140 // Deklarationsdel
0150 // Procedurer
0160
0170 PROC stol
0180     moveto(320,200)
0190     drawto(320,230)
0200     drawto(350,230)
0210     moveto(350,270)
0220     drawto(350,200)
0230 ENDPROC stol
```

Sid 101:

```

0010 // Huvudprogram
0020 INPUT "Ange främre stolsbenets placering, x0,y0: ": x0#,y0#
0030 INPUT "Ange benlängd: ": ben#
0040 INPUT "Ange sitsens bredd: ": sits#
0050 INPUT "Ange ryggstödet höjd: ": rygg#
0060 USE graphics // nyttritblock Behövs ej
0070 stol(x0#,y0#,ben#,sits#,rygg#)
0080
0090 // Deklarationsdel
0100 // Procedurer
0160
0170 PROC stol(x0#,y0#,ben#,sits#,rygg#)
0180   moveto(x0#,y0#)
0190   draw(0,ben#)
0200   draw(sits#,0)
0210   move(0,rygg#)
0220   draw(0,-rygg#-ben#)
0230 ENDPROC stol

```

Sid 102:

```

0010 // Huvudprogram
0020 USE graphics // nyttritblock - Behövs inte här
0030 placerabord
0040 placerastol
0050
0060 // Deklarationsdel
0070 // Procedurer
0130
0140 PROC placerabord
0150   INPUT AT 1,1: "Ange vänster bordskantsplacering, x0,y0: ":x0#,y0#
0160   INPUT AT 2,1: "Ange benlängd: ": ben#
0170   INPUT AT 3,1: "Ange kant (överhäng): ": kant#
0180   INPUT AT 4,1: "Ange skivans längd: ": skiva#
0190   bord(x0#,y0#,ben#,kant#,skiva#)
0200 ENDPROC placerabord
0210
0220 PROC bord(x0#,y0#,ben#,kant#,skiva#)
0230   moveto(x0#,y0#)
0240   draw(skiva#,0)
0250   moveto(x0#+kant#,y0#)
0260   draw(0,-ben#)
0270   moveto(x0#+skiva#-kant#,y0#)
0280   draw(0,-ben#)
0290 ENDPROC bord
0300
0310 PROC placerastol
0320   INPUT AT 1,1: "Ange främre stolsbensplacering, x0,y0: ": x0#,y0#
0330   INPUT AT 2,1: "Ange benlängd: ": ben#
0340   INPUT AT 3,1: "Ange sitsens bredd: ": sits#
0350   INPUT AT 4,1: "Ange ryggstödet höjd: ": rygg#
0360   stol(x0#,y0#,ben#,sits#,rygg#)
0370 ENDPROC placerastol
0380

```

```
0390 PROC stol(x0#,y0#,ben#,sits#,rygg#)
0400   moveto(x0#,y0#)
0410   draw(0,ben#)
0420   draw(sits#,0)
0430   move(0,rygg#)
0440   draw(0,-rygg#-ben#)
0450 ENDPROC stol
```

Sid 103:

```
1000 PROC övrehalv(x#,y#,r#)
1010   arc(x#,y#,r#,0,180)
1020 ENDPROC övrehalv
1030
1040 PROC undrehalv(x#,y#,r#)
1050   arc(x#,y#,r#,0,-180)
1060 ENDPROC undrehalv
1070
1080 PROC vänsterhalv(x#,y#,r#)
1090   arc(x#,y#,r#,90,180)
1100 ENDPROC vänsterhalv
1110
1120 PROC högerhalv(x#,y#,r#)
1130   arc(x#,y#,r#,-90,180)
1140 ENDPROC högerhalv
```

Sid 104:

```
0010 // Halvcirkel
0020 // Huvudprogram
0025 USE graphics
0030 window(0,639,0,399) // nytttritblock BEHÖVS EJ
0040 INPUT "Halvcirkelns medelpunkt x,y: ": x#,y#
0050 INPUT "Halvcirkelns radie r: ": r#
0060 övrehalv(x#,y#,r#)
0070
0080 // Deklarationsdel
0090 // Procedurer
0100
0150 PROC övrehalv(x#,y#,r#)
0160   arc(x#,y#,r#,0,180)
0170 ENDPROC övrehalv
```

Sid 105 (1:a ex):

```
0010 USE graphics
0020 window(0,639,0,399) // Ger samma koordsystem som Compis
0030 cylinder(320,200,50,25)
0999
1000 PROC cylinder(x#,y#,r#,h#)
1010   moveto(x#,y#)
1020   setheading(-45)
1030   arcl(r#,90)
1040   left(90)
1050   arcl(r#,90)
1060   left(45)
1070   forward(h#)
1080   left(45)
1090   arcl(r#,90)
1100   left(45)
1110   forward(h#)
1120 ENDPROC cylinder
```

Sid 105 (2:a ex):

```
0010 USE graphics
0015 window(0,639,0,399) // Ger samma koordsystem som Compis
0020 måne(500,250)
1000 PROC måne(x#,y#)
1010   moveto(x#,y#)
1020   setheading(45)
1030   arcl(50,120)
1040   left(155)
1050   arcr(75,70)
1060 ENDPROC måne
```

Sid 106:

```

0010 // Huvudprogram
0015 USE graphics
0020 window(0,639,0,399) // nytttritblock BEHÖVS EJ
0030 sjusegment(300,200,50)
0040
0100 // Deklarationsdel
0105 // Procedurer
0110
0120
0130 PROC sjusegment(x0#,y0#,b#)
0140   moveto(x0#,y0#)
0150   // Segment 1
0160   move(5,0)
0170   draw(b#-10,0)
0180   // Segment 2
0190   moveto(x0#+b#,y0#+5)
0200   draw(0,b#-10)
0210   // Segment 3
0220   move(0,10)
0230   draw(0,b#-10)
0240   // Segment 4
0250   moveto(x0#+b#-5,y0#+2*b#)
0260   draw(-b#+10,0)
0270   // Segment 5
0280   moveto(x0#,y0#+2*b#-5)
0290   draw(0,-b#+10)
0300   // Segment 6
0310   moveto(x0#+b#-5,y0#+b#)
0320   draw(-b#+10,0)
0330   // Segment 7
0340   moveto(x0#,y0#+b#-5)
0350   draw(0,-b#+10)
0360 ENDPROC sjusegment

```

Sid 107:

```

1000 PROC stapel(x#,y#,b#,h#)
1010   rectfill(x#,y#,x#+b#,y#+h#)
1020 ENDPROC stapel
1030
1050 // rectfill saknas i UniCOMAL
1060 PROC rectfill(x1,y1,x2,y2)
1070   clip(x1,x2,y1,y2)
1080   fill((x1+x2)/2,(y1+y2)/2)
1090   noclip
1100 ENDPROC rectfill

```

Sid 108 (1:a ex):

```

1000 PROC sektor(x#,y#,r#,v#,p)
1010   moveto(x#,y#)
1020   setheading(v#)
1030   forward(r#)
1040   delta#:=INT(p*3.6)
1050   arc(x#,y#,r#,v#,delta#)
1060   setheading(v#+delta#)
1070   moveto(x#,y#)
1080   forward(r#)
1090 ENDPROC sektor

```

Sid 108 (2:a ex):

```

0002 USE graphics
0010 // Cirkeldiagram
0020 // Huvudprogram
0030 window(0,639,0,399) // nyttritblock behövs ej
0035 showturtle // Visa sköldpaddan
0040 // Medelpunkt
0050 x#:=320
0060 y#:=200
0070 // Radie och utgångsvinkel
0080 r#:=100
0090 v#:=0
0100 INPUT "Antal sektorer: ": antal#
0110 FOR i#:=1 TO antal# DO
0120   INPUT "Procentsats: ": p
0130   sektor(x#,y#,r#,v#,p)
0140   v#:=v#+INT(3.6*p)
0150 ENDFOR i#
0160
0170 // Deklarationsdel
0180 // Procedurer
0190
0250 PROC sektor(x#,y#,r#,v#,p)
0260   moveto(x#,y#)
0270   setheading(90-v#)
0280   forward(r#)
0290   delta#:=INT(p*3.6)
0300   arc(x#,y#,r#,v#,delta#)
0310   setheading(90-(v#+delta#))
0320   moveto(x#,y#)
0330   forward(r#)
0340 ENDPROC sektor

```

Sid 110 (med testprog):

```
0010 USE graphics
0012 numcolors#:=16 // Gäller EGA o VGA. Måste ändras för andra
0015 window(0,639,0,399) // Ger samma koordsystem som Compis
0020 bort(200,50,200,200)
0025 FOR i:=1 TO 1000 DO NULL
0030 hit(400,100,40,40)
0040
0050 PROC bort(x0#,y0#,l#,h#)
0060   pencolor(2*numcolors#-1) // = penmode(1)
0070   FOR i#:=1 TO 20 DO
0080     rectangle(x0#,y0#,x0#+l#,y0#+h#) // anv. egen procedur
0090     x1#:=x0#; y1#:=y0#; l1#:=l#; h1#:=h#
0100     x0#:=x0#+10
0110     y0#:=y0#+2
0120     l#:=INT(0.9*l#)
0130     h#:=INT(0.9*h#)
0140     rectangle(x1#,y1#,x1#+l1#,y1#+h1#)
0150   ENDFOR i#
0160 ENDPROC bort
0170
0180
0190 PROC hit(x0#,y0#,l#,h#)
0200   pencolor(2*numcolors#-1) // = penmode(1)
0210   FOR i#:=1 TO 20 DO
0220     rectangle(x0#,y0#,x0#+l#,y0#+h#)
0230     x1#:=x0#; y1#:=y0#; l1#:=l#; h1#:=h#
0240     x0#:=x0#-10
0250     y0#:=y0#-2
0260     l#:=INT(1.1*l#)
0270     h#:=INT(1.1*h#)
0280     rectangle(x1#,y1#,x1#+l1#,y1#+h1#)
0290   ENDFOR i#
0300 ENDPROC hit
0310
0320 PROC rectangle(x1#,y1#,x2#,y2#)
0330   moveto(x1#,y1#)
0340   drawto(x2#,y1#)
0350   drawto(x2#,y2#)
0360   drawto(x1#,y2#)
0370   drawto(x1#,y1#)
0380 ENDPROC rectangle
```

Kapitel 3

Rättelser

OBS: I vissa bokexempel föregås det som ska skrivas ut med PRINT AT av semikolon (;). I UniCOMAL ska det *alltid* vara kolon (:), dvs det heter **PRINT AT rad#,kol#:** ... Detta kommer inte att påpekas i fortsättningen.

Sid 143 (mitten):

Programkörning stoppas med *Ctrl-Break*.

Sid 148:

I början av avsnittet om funktioner står att ”en funktion har en variabel, formell parameter ...”. Detta är inte helt korrekt. Det som utmärker funktioner är att de returnerar ett värde via sitt namn. Antalet parametrar kan däremot variera: Funktioner kan såväl helt sakna parametrar som ha en, två eller flera. Exempel på detta ges senare i kapitlet (sid 187 ff).

Sid 168:

I PRINT-satser kan nyckelordet **AT** enbart användas i direkt anslutning till PRINT (dvs som PRINT AT). AT som ligger mellan utskriftssträngar ersätts lämpligen med TAB-funktionen. Gör följande ändringar i det längre programexemplet:

```
40 PRINT AT 3,5:"Artikel";TAB(20);"Artikelnummer";TAB(37);"Lagervärde"
110 PRINT AT rad#,5: artnamn$;TAB(20);artnr$
120 PRINT AT rad#,37: USING "#####.##": lagervärde
```

Sid 169:

Se ovan.

Sid 174:

Programexemplet på mitten av sidan är inte helt korrekt (varför?). Bättre (och korrekt) lösning:

```
0010 // Dimensionering
0020 DIM tal(25)
0030 // Inmatning
0040 antal#:=0
0050 INPUT "Ange ett tal (-999 för att sluta): ": temp
0055 WHILE temp <> -999 AND antal# < 25 DO
0060   antal#:=antal#+1
0065   tal(antal#):=temp
0070   INPUT "Ange ett tal (-999 för att sluta): ": temp
0080 ENDWHILE
0090
0100 // Bearbetning
0110 min:=tal(1)
0120 FOR i#:=2 TO antal# DO
0130   IF tal(i#)<min THEN min:=tal(i#)
0140 ENDFOR i#
0150 // Utmatning
0160 PRINT "Minsta talet =" ;min
```

Sid 178:

Ändra följande programrader:

```
200 PRINT AT 3,5: "Bokstav";TAB(17);"Frekvens"
230 PRINT AT rad#,8: alfabete$(j#:j#);TAB(20);frekvens(j#)
260 PRINT AT 3,35: "Bokstav";TAB(47);"Frekvens"
290 PRINT AT rad#,38: alfabete$(j#:j#);TAB(50);frekvens(j#)
```

Sid 186:

Korrigerade följande programrader:

```
200 PRINT AT 3,5:"Artikel";TAB(20);"Artikelnr";TAB(37);"Lagervärde"
290 PRINT AT rad#,5: artnamn$;TAB(20);artnr$
300 PRINT AT rad#,37: USING "#####.##": lagervärde
```

Sid 189:

Exemplet längst ned på sidan: De svenska bokstäverna ÅÄÖ följer inte direkt efter bokstaven Z med den ASCII-kod som används med IBM-kompatibla datorer (se ASCII-tabellen i slutet av kompendiet). Därför får vi inskränka oss till att skriva ut A-Z, dvs byt ut talet 28 mot 25 i FOR-satsen.

VAL

Sid 190:

PRINT VAL("123XYZ") ger inte svaret 123, utan ett felmeddelande: **Argumentfel**. Orsaken är att strängen innehåller tecken som inte kan ingå i ett tal.

Sid 192:

KEY\$-funktionen ger en tom sträng om ingen tangent nedtryckts och eftersom en tom sträng inte kan ha något ORD-värde fungerar inte 1:a och 3:e exemplet på sidan. Det första kan i stället skrivas enligt modellen i 2:a exemplet:

```
0005 DIM tkn$ OF 1
0010 PAGE
0020 PRINT "Stopp i programmet. Tryck <C> eller <c> för forts."
0030 REPEAT
0040   tkn$:=KEY$
0050 UNTIL tkn$ IN "Cc"
0060 PRINT "Programmet fortsätter med denna sats."
```

Det sista (3:e) exemplet kan skrivas så här:

```
0005 DIM tkn$ OF 1
0010 PAGE
0020 PRINT "Stopp i programmet. Tryck valfri tangent för forts."
0030 REPEAT
0040   tkn$:=KEY$
0050 UNTIL tkn$<>" "
0060 PRINT "Programmet fortsätter med denna sats."
```

Mera om villkorssatser (Tillägg sid 141)

Läroboken behandlar 3 former av IF-satsen:

1. Enkel (enradig) IF-sats:

```
IF <villkorsuttryck> THEN <sats>
```

Exempel:

```
IF svar$ = "J" THEN PRINT "Ja!!"
```

Den enradiga IF-satsen kan endast användas i de allra enklaste fallen.

2. Flerradig IF-sats:

```
IF <villkorsuttryck> THEN
  <satser>
ENDIF
```

Exempel:

```
IF tal<>0 THEN
  inverterat_tal:=1/tal
  PRINT "Det inverterade talet =" ;inverterat_tal
ENDIF
```

3. Förgrenad IF-sats

```
IF <villkorsuttryck> THEN
  <satser>
ELSE
  <satser>
ENDIF
```

Exempel:

```
IF svar$ = "J" THEN
  PRINT "Ja!!"
ELSE
  PRINT "Nej!"
ENDIF
```

Det finns ytterligare en IF-variant, med vilken flera villkor efter varandra kan testas:ELIF

```
IF <villkorsuttryck1> THEN
  <satser>
ELIF <villkorsuttryck2> THEN
  <satser>
ELSE
  <satser>
ENDIF
```

Det nya är nyckelordet **ELIF**. Detta är en sammandragning av de två orden ELSE IF som kan översättas med "annars om".

I konstruktionen kan ingå ett godtyckligt antal ELIF-delar. ELSE-delen är ej obligatorisk, dvs kan uteslutas.

Exempel

Antag att vi vill jämföra två inmatade tal. Om talen är lika ska detta meddelas, i annat fall ska det största bestämmas och skrivs ut. Lösningförslag i halvkod (pseudokod):

```
inmata talen
om talen är lika
  meddela detta
annars om tal1 > tal2
  skriv ut tal1
annars
  skriv ut tal2
```

Programkod:

```
INPUT "Ange 2 tal: ":tal1,tal2
IF tal1=tal2 THEN
  PRINT "Talen är lika"
ELIF tal1>tal2 THEN
  PRINT tal1;"är störst"
ELSE
  PRINT tal2;"är störst"
ENDIF
```

ELIF förenklar hanteringen av flervalssituationer, där man annars skulle tvingas att ta till tillkrånglade "nästlade" IF-satser, dvs IF-satser inuti andra IF-satser. Du kommer att träffa på åtskilliga problem där ELIF-varianten är bekväm och lämplig att använda.

Anmärkning: Motsvarigheter till ELIF finns även i andra moderna programspråk, såsom Ada och Modula-2.

Övningsuppgift

3-1. Skriv om läroboksexemplet på sid 141 med ELIF.

Av lärobokens övningsuppgifter är bl a följande "ELIF-kandidater": 3.28, 3.31, 3.32 och 3.50.

Boolska datatyper – **TRUE/FALSE** (Tillägg sid 146)

I COMAL finns två fördefinierade ”symboliska” konstanter, **FALSE** och **TRUE** (”falskt” och ”sant”) som är användbara när man har att göra med villkor av olika slag.

FALSE och TRUE representeras internt av heltalsvärdena 0 resp 1. Detta behöver man för det mesta inte tänka på, utan de kan behandlas som ”logiska konstanter”, det sanna och det falska värdet.

Variabler som kan anta värdena TRUE och FALSE brukar kallas *boolska*. Vissa programspråk, såsom Pascal, har en speciell *boolsk datatyp* och variabler av denna typ kan enbart anta boolska värden (dvs FALSE och TRUE), precis som heltalsvariabler bara kan anta heltalsvärden. COMAL saknar en boolsk datatyp, men vi kan utnyttja numeriska typer, dvs talvariabler, för att representera ”boolska värden”.

Talvariabler kan tilldelas boolska konstanter:

```
slut :=FALSE
sant#:=TRUE
stor#:=TRUE
hemma:=FALSE
```

Däremot kan inte en variabel ges värdet TRUE eller FALSE genom att inmata *ordet* TRUE eller FALSE via en INPUT-sats.

Boolska variabler och konstanter är ofta naturliga att använda i villkorsuttryck och kan göra programmen lättare att överblicka och förstå.

Exempel

I ett program vill vi göra det möjligt att välja om utskrift ska ske på bildskärmen eller på skrivare. Detta kan ”styras” med en boolsk variabel, `på_skrivare`:

```
DIM svar$ OF 1

INPUT "Vill du ha utskriften på skrivare? ": svar$
IF svar$="J" OR svar$="j" THEN
  på_skrivare:=TRUE
ELSE
  på_skrivare:=FALSE
ENDIF
```

På annan plats i programmet kan variabeln `på_skrivare` utnyttjas för att styra utskriften dit vi vill ha den. Detta kan göras så här:

```
IF på_skrivare=TRUE THEN SELECT OUTPUT "PRN:"
```

men vi kan också helt enkelt skriva

```
IF på_skrivare THEN SELECT OUTPUT "PRN:"
```

vilket gör programmet ännu mera lättläst.

Exempel

Antag att vi vill förhindra att kreti och pleti ska kunna köra ett visst program. Vi kan då avkräva användaren ett lösenord innan han/hon anses vara ”behörig” och får fortsätta programkörningen:

```
DIM lösenord$ OF 10, svar$ OF 10

lösenord$:="hemligt"
INPUT "Skriv lösenordet: ": svar$
IF svar$=lösenord$ THEN
  behörig:=TRUE
ELSE
  behörig:=FALSE
ENDIF
IF behörig THEN
  PRINT "Välkommen"
ELSE
  PRINT "Du är inte behörig"
  END // Programkörningen avbryts
ENDIF
// Programmet fortsätter ...
```

Studera första IF-ELSE-satsen. Där erhåller behörig värdet TRUE eller FALSE beroende på om rätt lösenord inmatats eller inte. Dessa programrader kan faktiskt ersättas med en enda:

```
behörig:=svar$=lösenord$
```

Tänk igenom ovanstående sats ordentligt. Efter tilldelningsoperatoren följer villkorsuttrycket svar\$=lösenord\$. Detta kan vara sant (om svar\$ är lika med lösenord\$) eller falskt (svar\$ är *inte* lika med lösenord\$). Variabeln behörig tilldelas värdet av detta villkorsuttryck, dvs får värdet TRUE eller FALSE.

Exempel

Betrakta villkorsuttrycket

```
sök$ IN sträng$
```

Uttrycket får värdet FALSE (=0) om sök\$ *inte* finns i sträng\$. I annat fall erhålls positionen för sök\$, dvs ett tal > 0. Detta hanterar COMAL som ett sant värde (TRUE).

Följande exempel illustrerar det hela:

```
DIM siffror$ OF 10, tkn$ OF 1

siffror$:="0123456789"
INPUT "Skriv ett tecken: ": tkn$
siffra:=tkn$ IN siffror$
IF siffra THEN
  PRINT "En siffra!"
ELSE
  PRINT "Det var ingen siffra."
ENDIF
```

Boolska variabler kan också användas i villkorsuttryck där de logiska operatorerna (AND, OR och NOT) ingår.

Dessa operatörer fungerar enligt följande:

uttryck1 AND uttryck2	SANT om både uttryck1 och uttryck2 är sanna
uttryck1 OR uttryck2	SANT om uttryck1 eller uttryck2 är sant
NOT uttryck3	Negerar det logiska uttrycket uttryck3. Om uttryck3 är falskt blir NOT uttryck3 sant och tvärtom.

Exempel

De logiska operatorernas funktion kan studeras med följande program:

```
DIM s$ OF 1

INPUT "Ange a:s sanningsvärde, S=sant, F=falskt: ": s$
a:=s$ IN "Ss"
INPUT "Ange b:s sanningsvärde, S=sant, F=falskt: ": s$
b:=s$ IN "Ss"
IF a AND b THEN
  PRINT "a AND b = sant"
ELSE
  PRINT "a AND b = falskt"
ENDIF
```

Övningsuppgifter

3-2. Lös läroboksuppgift 3.9a med hjälp av en boolsk variabel, `vokal`, som ges värdet `TRUE` (sant) om bokstaven är en vokal och `FALSE` annars.

(Andra läroboksuppgifter där boolska variabler kan vara lämpliga är 3.16, 3.29 och 3.62)

3-3. Undersök de logiska operatorerna `OR` och `NOT` med program liknande det som behandlar `AND`-operatören ovan. Pröva gärna också kombinationer av logiska operatörer, såsom `a AND NOT b`.

3-4. Skriv ett program som kräver inmatning av ett tal mellan 0 och 1 och kontrollerar att det inmatade talet håller sig inom dessa gränser. Låt programmet ge lämpliga meddelanden beroende på om allt är ok eller inte. Utnyttja en boolsk variabel, `korrekt`, för detta ändamål.

Operatorerna DIV och MOD (Tillägg sid 152)

Förutom de "vanliga" aritmetiska operatorerna +, -, *, / och ^ finns i COMAL även DIV och MOD, vilka används vid beräkningar med heltal:

a DIV b ger heltalskvoten när a divideras med b
a MOD b ger resten när a divideras med b

Exempel:

9 DIV 3 är 3 och 9 MOD 3 är 0, ty $9/3 = 3$ med resten 0
1 DIV 2 är 0 och 1 MOD 2 är 1, ty $1/2 = 0$ med resten 1
7 DIV 5 är 1 och 7 MOD 5 är 2, ty $7/5 = 1$ med resten 2

DIV och MOD är definierade för heltal och ger heltal som resultat. I motsats till andra program-språk kontrollerar dock inte COMAL detta, varför operatorerna kan användas ihop med godtyckliga numeriska datatyper. Tag dock för vana att utnyttja dem enbart för heltal.

DIV och MOD har samma prioritet som * och /.

Vilket existensberättigande har DIV och MOD? Räcker det inte med vanlig division? Jo, det finns faktiskt många tillfällen där DIV och MOD är att föredra.

Vid vanlig division blir resultatet (kvoten) alltid ett *flyttal* (med decimaler om divisionen inte går jämnt upp). När vi arbetar med *heltal* och bara är intresserade av heltalsresultat, såsom hur många hela gånger ett tal går i ett annat och/eller hur stor heltalsresten blir vid en division, då bör DIV och MOD användas i stället för vanlig division, ty de ger heltal till resultat (dvs inga avrundningsfel) och exekverar mycket snabbare än vanlig division.

Låt oss titta på några exempel!

Delbarhet

I läroboken (sid 151) används INT-funktionen för att avgöra delbarhet. För att ett tal ska vara jämnt delbart med t ex 5, måste nämligen gälla att $\text{INT}(\text{tal}\#/5) = \text{tal}\#/5$, dvs kvoten får ej innehålla decimaler. Alternativt kan man här utnyttja MOD-operatoren, ty MOD ger resten vid heltalsdivision, och divisionen måste ju gå jämnt upp om resten är 0.

Uttrycket

125 MOD 5=0

är sant, vilket betyder att 125 är jämnt delbart med 5, men uttrycket

126 MOD 5=0

är *inte* sant, dvs 126 är inte delbart med 5.

Följande program kontrollerar om ett tal är jämnt eller udda:

```
INPUT "Ange ett tal: ": tal#
IF tal# MOD 2=0 THEN
  PRINT tal#;"är jämnt."
ELSE
  PRINT tal#;"är udda."
ENDIF
```

Uppacka siffror (jfr boken sid 151-152)

Om vi skriver

```
heltal#:=4721
PRINT heltal# MOD 10
```

svarar datorn:

1

dvs vi erhåller entalssiffran.

Skriver vi

```
PRINT heltal# DIV 10
```

svarar datorn:

472

dvs övriga siffror.

Med upprepad användning av DIV och MOD kan alla siffrorna skiljas ut. För att ”packa upp” heltalet 4721 skulle vi t ex kunna göra så här:

```
heltal#:=4721
entalssiffra#:=heltal# MOD 10      // Ger 1
heltal#:=heltal# DIV 10           // Ger 472
tiotalssiffra#:=heltal# MOD 10    // Ger 2
heltal#:=heltal# DIV 10           // Ger 47
hundratalssiffra#:=heltal# MOD 10 // Ger 7
heltal#:=heltal# DIV 10           // Ger 4
tusentalssiffra#:=heltal# MOD 10  // Ger 4
```

Sidbyte

Vid utskrift av långa listor vill man ofta göra sidbyte efter ett visst antal rader. För detta kan vi utnyttja MOD-operatorn.

Antag att vi vill ha 20 rader/sida. Om vi med variabeln radantal# håller räkning på antalet utskrivna rader kan sidbytena hanteras med satsen:

```
IF radantal# MOD 20=0 THEN PAGE // Sidbyte
```

Växelpengar

Följande program är en ”automatisk kassaapparat”. Det omvandlar återbetalningsbeloppet (i jämna kr) till antal sedlar och mynt:

```
INPUT "Varans pris (kr)      : ": pris#
INPUT "Emottaget belopp (kr): ": belopp#
PRINT
åter#:=belopp#-pris#
PRINT "Att betala tillbaka:"
PRINT åter# DIV 100;"hundrakronorssedlar"
åter#:=åter# MOD 100
```

```

PRINT åter# DIV 50;"femtiokronorssedel"
åter#:=åter# MOD 50
PRINT åter# DIV 10;"tiokronorssedlar"
åter#:=åter# MOD 10
PRINT åter# DIV 5;"femkrona"
åter#:=åter# MOD 5
PRINT åter#;"enkronor"

```

Övningsuppgifter

3-5. Bestäm värdena av följande uttryck. Försök lösa uppgifterna utan datorhjälp:

- | | |
|-------------|--------------|
| a) 1 DIV 3 | e) 97 DIV 2 |
| b) 1 MOD 3 | f) 97 MOD 2 |
| c) 16 DIV 4 | g) 55 DIV 10 |
| d) 16 MOD 4 | h) 55 MOD 10 |

3-6. Bestäm värdena av uttrycken

- | | |
|-------------------|------------------|
| a) 2 DIV 3 + 3/5 | d) 7 MOD 5 MOD 3 |
| b) (2+3) MOD 2 | e) (7*5 MOD 3)*4 |
| c) 17 DIV 5 MOD 3 | f) 25*(5 DIV 2) |

3-7. I en uppslagsbok står följande att läsa om skottår:

”Skottår, år som har skottdag och därför 366 dagar i stället för 365. Skottår är alla år vilkas årtal är jämnt delbara med 4, utom de sekelår vilkas sekeltal inte är jämnt delbara med 4 (t ex 1800, 1900).”

Utgå från beskrivningen och formulera en programsats som, med årtalet (ett heltal) givet, ger skottår värdet TRUE, om året är ett skottår och FALSE annars. Skriv sedan en INPUT-sats och testa det hela med ovanstående årtal (är inte skottår) och några andra, fritt valda. Utskriften kan t ex göras med en sats av typen

```

IF skottår THEN
  PRINT "Skottår"
ELSE
  PRINT "Ej skottår"
ENDIF

```

3-8. Med DIV och MOD kan kvoten av två heltal beräknas med hur stor noggrannhet som helst. Tekniken är densamma som du använder när du dividerar för hand: Först bestäms heltalskvoten och resultatet skrivs upp. Sedan lägger man till en nolla till resten och flyttar decimalkommat ett steg åt höger (vad innebär det matematiskt?) och kvoten mellan detta nya heltal och nämnaren bestäms (ger 1:a decimalen). Detta upprepas tills önskat antal decimaler erhållits. (Det är lämpligt att utföra en sådan division ”för hand” för att kontrollera att det sagda stämmer).

Skriv ett program som frågar efter täljaren, nämnaren och antalet decimaler och sedan beräknar kvoten med denna noggrannhet. Utmata svaret i lämpligt format.

I kapitel 3 i läroboken finns flera uppgifter som är lämpliga att lösa med DIV och/eller MOD: 16, 17, 20b, 25, 38, 45.

Mera om upprepningssatser – LOOP-~~END~~LOOP (Tillägg sid 163)

I COMAL finns ytterligare en upprepningssats: LOOP-satsen.

Konstruktionen ser ut så här

```
LOOP
  <satser>
ENDLOOP
```

LOOP-satsen ger en s k villkorlös upprepning, dvs upprepningvillkor anges inte. Upprepningen blir ”oändlig” om vi inte *inne i* slingan anger något stoppvillkor, där uthoppet åstadkoms med en EXIT-sats. Denna kan skrivas som en ensam sats eller tillsammans med ett villkor, t ex EXIT WHEN <villkor>:

```
LOOP
  <satser>
  EXIT WHEN <villkor>
  <satser>
ENDLOOP
```

Uthoppsvillkoret kan även formuleras med en IF-sats:

```
IF <villkor> THEN EXIT
```

LOOP-strukturen är en mycket kraftfull upprepningssats som bör brukas med måtta. För det mesta klarar man sig utmärkt med REPEAT-UNTIL och WHILE-ENDWHILE, vilka har den fördelen att avbrottsvillkoren resp upprepningvillkoren har en fast ”position” (sist resp först) i slingstrukturen, medan uthoppet i LOOP-~~END~~LOOP kan ske var som helst inne i slingan.

Ett exempel på hur LOOP-satsen fungerar ska vi dock ge:

```
LOOP
  INPUT "Skriv ett tal: ": tal
  EXIT WHEN tal=0
  PRINT "Du skrev";tal
ENDLOOP
```

Talinmatningen och utskriften "Du skrev..." upprepas tills dess användaren skrivit 0, varvid slingan lämnas.

Flera EXIT kan förekomma i en LOOP-slinga, men för det mesta är det dålig programmering att utnyttja den möjligheten: Det bör bara finnas ”en utgång”.

Sökning (Tillägg sid 186)

Vid hantering av lite större datamängder behövs ofta sökningar göras för att finna vissa data. Vid sökning är det vanligt att använda en boolsk variabel som en ”flagga”, som talar om huruvida det som söks har hittats eller inte. Följande programexempel, som demonstrerar s k *linjär sökning* (elementen gås igenom ett efter ett), får illustrera tekniken:

```

DIM månad$ OF 3, svar$ OF 3

REPEAT
  INPUT "Skriv en månads namn (tre bokstäver ): ": svar$
  funnet:=FALSE
  RESTORE
  WHILE NOT funnet AND NOT EOD DO
    READ månad$, dag
    IF månad$=svar$ THEN funnet:=TRUE
  ENDWHILE
  IF funnet THEN
    PRINT månad$;"har";dag;"dagar."
  ELSE
    PRINT "Denna månad finns ej !"
  ENDIF
UNTIL FALSE // Oändlig loop
DATA "jan",31,"feb",28,"mar",31,"apr",30,"maj",31,"jun",30
DATA "jul",31,"aug",31,"sep",30,"okt",31,"nov",30,"dec",31

```

Kapitel 4

Rättelser

Sid 210:

Byt ut "printer" (mitten på sidan) mot "prn: ".

Sid 210:

Inte heller med vår COMAL behöver man skriva in **FOUND**-kommandot för hand. Man kan nedtrycka funktionstangenten **F9** i stället.

Sid 211:

När sökningen är avslutad erhålls ingen speciell utskrift (stryk "END OF FIND")

Sid 211:

I UniCOMAL används **FOUND** enbart för sökning. För utbyte används **CHANGE**-kommandot. För att text byta ut variabeln `n#` mot `antal#` skriver vi

```
CHANGE "n#","antal#"
```

Hela programmet genomsöks och förekomsterna visas på skärmen. Om utbyte önskas trycks ↵ och om inte trycks **N**-tangenten. I båda fallen fortsätter sökningen med nästa förekomst. Sökningen/utbytet kan avbrytas med **Ctrl-Break**.

Sid 214:

Byt ut satserna `PRINT AT rad#,kol#;` mot `CURSOR rad#,kol#`

Sid 216:

Programkod kan sparas på skiva i ASCII-format *utan* radnummer. Det görs med kommandot

```
DISPLAY "<filnamn>"
```

Även *delar av* ett program och procedurer/funktioner kan sparas på detta sätt, precis som med **LIST**-kommandot.

Programkod kan också listas direkt till skrivare (`DISPLAY "PRN: "`).

Användning av **DISPLAY** i stället för **LIST** rekommenderas, ty listningarna blir snyggare utan radnummer (dessutom sparas diskutrymme).

Sid 217:

Kommandot **ENTER** raderar det program som finns i minnet, dvs fungerar i detta avseende som **LOAD**. För att sammanfoga programavsnitt måste **MERGE** användas. Observera att det inte spelar någon roll om programkoden sparas med **LIST** eller **DISPLAY** (det fungerar likadant i båda fallen).

Sid 218-219:Grafik

I bokens programexempel utnyttjas grafik, vilken ofta skiljer sig väsentligt från en dator till en annan. Nedan ges de ändringar som behövs för att programmet ska fungera med en PC:

- 1) Tillfoga satsen **USE graphics** först i programmet
- 2) I proceduren `nytttritblock` ska stå `graphicscreen(0)` (ett "s"). Har din dator EGA/VGA-kort kan du helt enkelt stryka satsen.
- 3) Tillfoga `window(0,639,0,399)` sist i proceduren `nytttritblock`.
- 4) Skriv till följande procedur:

```
PROC rectfill(x1,y1,x2,y2)
  clip(x1,x2,y1,y2)
  fill((x1+x2)/2,(y1+y2)/2)
  noclip
ENDPROC rectfill
```

Sid 221-222:Grafik

Återigen ett programexempel med grafik. Gör följande ändringar:

- 1) Tillfoga satsen `USE graphics` först i programmet
- 2) Ändra programrad 390 till `textstyle(1,1,0,1)`
- 3) Stryk programrad 490
- 4) Har datorn EGA- eller VGA-kort bör du stryka `graphicscreen(0)`
- 5) Tillför `window(0,639,0,399)` sist i proceduren `nytttritblock`

Sid 223:

I UniCOMAL (vår COMAL) behandlas styrvariabler i **FOR-loopar** automatiskt som *lokala* variabler, varför exemplet inte fungerar på det sätt som beskrivs i läroboken. Vill du studera effekten av *globala* variabler i underprogram kan du i stället studera exemplet i avsnittet om slutna procedurer senare i detta kompendium.

Sid 225:

Felmeddelandet (programkörningsexemplet) blir:

```
I 0110: tecken_nr#: Okänd variabel
```

Satsen `GLOBAL` finns ej i vår COMAL.

Sid 228:

I sista stycket (före exemplet) står att parenteser `()` inte ska anges för indexerade variabler vid anrop av proceduren `addera`. Detta stämmer inte för vår COMAL. Det aktuella avsnittet bör alltså lyda:

– Anropa proceduren med `addera(antal, nytt_tal, tal())`. Parenteser `()` ska alltså anges för indicerade variabler även i anropet.

Sid 229:

I enlighet med kommentaren ovan ska programrad 130 lyda:

```
130 addera(antal#,nytt_tal,tal())
```

Dessutom bör rad 150 ersättas med:

```
150 PAGE
```

Sid 234:

På mitten av sidan står: ”Parenteserna för den indicerade variabeln `tal()` skall ej anges vid anropet.”. Som påpekats ovan gäller detta inte UniCOMAL: Parenteserna *ska* anges. De aktuella programraderna i exemplet ska alltså se ut så här:

```
60 intal(antal_tal#,tal())
70 medelvärde(antal_tal#,tal(),medelv)
80 standardavvik(antal_tal#,tal(),medelv,standardavv)
```

Sid 235:

I första stycket står att en funktion inte kan returnera en indexerad variabel. Detta gäller inte vår COMAL: *En funktion kan returnera en godtycklig typ*, dvs även en indexerad variabel.

Lägg märke till att reglerna för funktionsnamn är desamma som för variabelnamn. Funktioner som returnerar heltalsvärden ska således ha tecknet `#` sist i funktionsnamnet.

Sid 238:

Följande kommentarer gäller funktionen `signatur` längst upp på sidan.

Tilllägg `CLOSED` sist i funktionshuvudet, ty funktionen måste vara sluten om den ska kunna anropas mer än en gång (en variabel i ett öppet underprogram får inte omdimensioneras).

Koderna 91-93 är inte bokstäver i IBM-ASCII. Därför kan inte **å-ö** konverteras till stora bokstäver med bokens metod, utan vi får nöja oss med att klara av **a-z**. Ändra därför programrad 160 till

```
160 IF x#>=65 AND x#<=90 THEN
```

Tecknet `#` skrivs *inte* efter ett tal i vår COMAL. Ändra programrad 170 till:

```
170 x#:=x#+32
```

Sid 240:

Ändra följande rader i sista exemplet (för att `ZONE` ska fungera OK):

```
70 PRINT AT 2,1: "x";"f(x)"
90 PRINT x;f(x)
```

Sid 241:

Korriger programrad 90 till:

```
90 PRINT "Summan =";summa(antal_tal#,tal())
```

Sid 243:

Ändra följande programrader till:

```
100 sum1:=summa(antal_tal#,tal())
110 sum2:=summakvad(antal_tal#,tal())
```

Sid 244:

Funktionen `siffersum` returnerar ett heltalsvärde och bör därför vara en heltalsfunktion, dvs tillfoga `#` sist i funktionsnamnet (ändra till `siffersum#(s$)`).

Sid 245-249:

I alla övningsexempel bör heltalsvariabler användas när så är möjligt. Detta gäller uppgifterna 4.9, 4.10, 4.21, 4.25, 4.28, 4.32, 4.33, 4.37-4.39, 4.42, 4.43.

Dessutom bör `TRUE`/`FALSE` returneras från alla funktioner där det är möjligt (och naturligt), dvs alla funktioner som undersöker om ett villkor är sant eller falskt. Gäller 4.31-4.33, 4.35-4.36, 4.44-4.46. Se vidare avsnittet om `TRUE` och `FALSE` i detta kompendium.

Sid 245:

Den efterfrågade proceduren i Övning 4.9 bör heta `upprepa_text(n#, t$)`.

Sid 246:

Rättelse (Övning 4.15): Ska stå `teckenrad(t$, n#)`.

Rättelse (Övning 4.23): Ska stå `striptext(REF t$)`.

Sid 249:

Första raden: Ändra `skylt$(rek s())` till `skylt$`, dvs stryk parenteserna.

Sid 249:

Hänvisningen i Övning 4.46 ska vara till 3.63 (ej 3.64). Funktionen bör även skrivas som en "boolsk funktion" (se ovan).

Sid 250:

Proceduren `intervallhalv` ska naturligtvis vara sluten. Funktionen görs tillgänglig genom `IMPORT`-deklaration, ej `GLOBAL` (som ej finns i vår `COMAL`).

Sid 251:

Stryk avsnittet om `GLOBAL`.

(Tillägg till funktionsavsnittet, börjar sid 235)

Boolska funktioner

Funktioner kan vara ”boolska”, dvs returnera värdena `FALSE` och `TRUE`.

Exempel

Ett tal är jämnt om det är jämnt delbart med 2. Följande funktion avgör om så är fallet:

```
FUNC jämnt(num#)
  IF num# MOD 2=0 THEN
    RETURN TRUE
  ELSE
    RETURN FALSE
  ENDIF
ENDFUNC jämnt
```

Funktionen kan skrivas ännu ”kompaktare”:

```
FUNC jämnt(n#)
  RETURN n# MOD 2=0
ENDFUNC jämnt
```

Funktionen kan t ex utnyttjas så här:

```
INPUT "Skriv ett heltal: ": tal#
IF jämnt(tal#) THEN
  PRINT "Talet är jämnt"
ELSE
  PRINT "Talet är udda"
ENDIF
```

För fullständighets skull ska vi här också visa ett annat sätt att utnyttja funktionen `jämnt` ovan. Det är nämligen så att `FALSE` och `TRUE` kan användas som index i en indexerad variabel! Om vi låter denna indexerade variabel innehålla strängarna ”udda” respektive ”jämnt” kan vi erhålla samma svar som ovan på följande sätt (vi tillåter nu inmatning av flera tal):

```
DIM typ$(FALSE:TRUE) OF 5

typ$(FALSE) := "udda"
typ$(TRUE) := "jämnt"
INPUT "Ange ett tal (sluta med 0) : ": tal#
WHILE tal#>0 DO
  PRINT "Talet är";typ$(jämnt(tal#))
  INPUT "Ange ett tal (sluta med 0) : ": tal#
ENDWHILE
PRINT "KLART !"
```

I läroboken bör övningsuppgifterna 31-33, 33, 35-40, 44-46 lösas med boolska funktioner/variabler.

I uppgifterna 25, 32, 33, 37-40 är det dessutom lämpligt att utnyttja `DIV` och/eller `MOD`.

Underprogram – en sammanfattning

När ett stort problem ska lösas är det mer eller mindre nödvändigt att bryta ner det i mindre, hanterbara delproblem och lösa dessa var för sig. Tillgång till underprogram (procedurer och funktioner) underlättar detta avsevärt: Dellösningarna namnges och sedan kan man, med vissa förbehåll, glömma alla detaljer: Det räcker med att veta vad underprogrammen heter och vad de gör.

Nedan görs en kortfattad genomgång av de viktiga procedur- och funktionsbegreppen. Avsnittet, som innehåller det väsentligaste, kan vara lämpligt att studera för att få en överblick, liksom för att snabbt friska upp minnet, när så behövs.

Procedurer

Procedurer måste definieras innan de kan användas.

Exempel

```
PROC rubrik
  text$:=" R U B R I K "
  PAGE
  PRINT
  FOR i#:=1 TO 34 DO PRINT "=",
  PRINT text$,
  FOR i#:=35 TO 80-LEN(text$) DO PRINT "=",
  PRINT
ENDPROC rubrik
```

I COMAL har en procedur följande uppbyggnad:

- Procedurbeskrivningen inleds med ett *procedurhuvud*, bestående av ordet **PROC**, följt av ett *procedurnamn* (rubrik ovan).
- Därefter följer en eller flera satser, som specificerar det som proceduren ska utföra. Detta kallas för *procedurkroppen*.
- Proceduren avslutas med **ENDPROC**, följt av procedurnamnet.

För att programsatserna i en procedur ska utföras måste proceduren ”anropas”. Detta görs genom att man skriver procedurens namn (ovan: rubrik).

Parameteröverföring

Ofta är det önskvärt att kunna överföra värden till en procedur. Detta åstadkoms med *parametrar*. I procedurhuvudet skrivs sådana inom parentes efter procedurnamnet.

Exempel

```
PROC skriv_tärningskast(antal#)
  FOR i#:=1 TO antal# DO
    PRINT RND(1,6);
  ENDFOR i#
  PRINT
ENDPROC skriv_tärningskast
```

Parametrarna i procedurhuvudet (`antal#` ovan) kallas *formella*. Dessa erhåller värden när proceduren anropas. Om proceduren ovan anropas med `skriv_tärningskast(10)`, kommer `antal#` att ersättas av värdet 10. Man säger att 10 är *aktuell* parameter.

Alla datatyper kan användas som parametrar, men de aktuella och formella parametrarna, dvs i anropande sats resp i procedurhuvudet, måste vara lika till antalet och parvis ha samma datatyp.

Slutna procedurer

Exempel

```
PROC kvadrattabell(max)
  x:=0
  REPEAT
    PRINT USING "###.#      #####.##":x,x*x
    x:=x+0.1
  UNTIL x>max
ENDPROC
```

Proceduren `kvadrattabell` beräknar och skriver ut kvadraten på talen `0 . . max`. I proceduren utnyttjas variabeln `x` för att hålla reda på vilka värden som skall beräknas och skrivas ut, `x` fungerar som en räknare. Om en variabel med samma namn (`x`) förekommer i andra delar av programmet, så kommer dess värde att ändras när proceduren utförs. Man säger att variabeln `x` är *global*. För det mesta vill man undvika sådana "sidoeffekter", dvs variabler inne i proceduren får ej av misstag påverka variabelvärden i andra programdelar. Detta kan vi åstadkomma genom att sist i procedurhuvudet skriva ordet **CLOSED**. Proceduren blir då "sluten" och alla variabler i proceduren blir *lokala*. Proceduren fungerar därefter som en "svart låda", dvs man behöver bara veta vad den utför, inte hur, vilket är en stor fördel vid skrivning av stora program.

Exempel

```
PROC kvadrattabell(max) CLOSED
  . . .
resten som ovan
```

Om man i en sluten procedur behöver komma åt variabler (och andra procedurer) från andra delar av ett program kan man använda **IMPORT**, följd av namnen på de variabler och procedurer som vi vill göra tillgängliga i den aktuella proceduren. Import av variabler bör dock undvikas (utnyttja parametrar!). **IMPORT**-satsen bör i första hand reserveras för att importera andra procedurer (och funktioner – se nedan) när detta är nödvändigt.

Slutna (stängda) procedurer rekommenderas. Överföring av data mellan olika delar av ett program sker bäst med parametrar. Härigenom blir procedurerna lättare att läsa och använda och man undviker att variabelvärden ändras av misstag (nackdelen är att man får skriva lite mer).

Referensanrop

Det slag av parametrar vi behandlat hittills har överfört data (värden) *till* underprogram. De kallas därför även *värdeparametrar* eller *inparametrar*.

Om en procedur framställer värden som behövs i andra delar av programmet, kan vi skriva nyckelordet **REF** framför namnet på den parameter detta gäller. Sådana parametrar kan återföra värden till den anropande satsen och kallas ibland *utparametrar* (värden "förs ut" från proceduren).

Exempel

```
PROC stora_bokstäver(REF t$) CLOSED
  DIM tkn$ OF 1

  FOR i#:=1 TO LEN(t$) DO
    tkn$:=t$(i#:i#)
    IF tkn$>="a" AND tkn$<="z" THEN
      t$(i#:i#):=CHR$(ORD(tkn$)-32)
    ENDIF
  ENDFOR i#
ENDPROC stora_bokstäver
```

Proceduren `stora_bokstäver` omvandlar små bokstäver (a-z) till stora. Observera att den formella utparametern, strängen `t$`, ej ska dimensioneras, ty den erhåller automatiskt rätt dimension vid proceduranropet.

Vid referensanrop måste den aktuella parametern vara en *variabel*, ty proceduren måste ha en variabel att skicka tillbaka det nya värdet till. Anropet `stora_bokstäver("kalle")` är alltså otillåtet. Om vi vill omvandla strängen ”kalle” måste den därför först tilldelas en variabel (t ex `namn$`) – anropet blir sedan `stora_bokstäver(namn$)`.

Indexerade variabler

För att specificera att en parameter är en indexerad variabel skrivs parenteser efter variabelnamnet, t ex `tabell()`. Detta gäller både för den aktuella och formella parametern.

Stora indexerade variabler bör ofta `REF`-deklarerars även om inga värden ska överföras tillbaka från proceduren. Detta därför att vid *värdeanrop* (ej `REF`) skapas en kopia (en extra upplaga) av variabeln. För icke-indexerade variabler spelar detta normalt ingen roll, men för indexerade variabler (och stora strängar) kan kopieringen kräva både onödigt mycket tid och minnesutrymme. Detta undviks om variabeln `REF`-deklarerars.

Exempel

```
PROC skrivtabell(REF a(), antal#) CLOSED
  FOR i#:=1 TO antal# DO PRINT a(i#)
ENDPROC
```

Proceduren skriver ut de `antal#` första talen i en tal-”tabell”.

Anrop, t ex: `skrivtabell(a(),10)`

Funktioner

Den viktigaste skillnaden mellan en procedur och en funktion är att funktioner alltid (via funktionsnamnet) returnerar ett värde till den anropande satsen.

I COMAL finns ett flertal inbyggda funktioner, men man kan även skapa egna. En funktion deklarerars enligt modellen:

```
FUNC <funktionsnamn>( <parameterlista> )
  <satser>
  RETURN <uttryck>
ENDFUNC <funktionsnamn>
```

RETURN-satsen ger funktionen dess värde och ”returnerar” detta från funktionen. När **RETURN**-satsen utförs återgår programexekveringen omedelbart till den anropande satsen, dvs eventuella programsatser *efter* **RETURN** utförs ej.

Funktioner kan både ge tal och strängar som resultat. Är funktionen av heltalstyp, markeras detta med tecknet **#** sist i funktionsnamnet. Är den av strängtyp, anges det med tecknet **\$**.

Det som tidigare sagts om parametrar, **REF**-parametrar, **CLOSED**, **IMPORT** osv i proceduravsnitten, gäller även funktioner.

En funktion används med fördel om bara *ett* variabelvärde ska återföras till anropande sats. Om flera variabelvärden, eller om inga värden alls ska återföras, då bör procedurer utnyttjas i stället.

Exempel

```
FUNC max(a,b) // Returnerar det största av talen a och b.
  IF a>b THEN
    RETURN a
  ELSE
    RETURN b
  ENDIF
ENDFUNC max
```

```
FUNC stora_bokstäver$(t$) CLOSED // Jfr proceduren ovan
  DIM tkn$ OF 1
  FOR i#:=1 TO LEN(t$) DO
    tkn$:=t$(i#:i#)
    IF tkn$>="a" AND tkn$<="z" THEN
      t$(i#:i#):=CHR$(ORD(tkn$)-32)
    ENDIF
  ENDFOR i#
  RETURN t$
ENDFUNC stora_bokstäver$
```

Eftersom en funktion returnerar ett värde genom sitt funktionsnamn kan funktionsanrop införas direkt i beräkningsuttryck och **PRINT**-satser, vilket inte är möjligt med procedurer. Man kan t ex skriva:

```
PRINT stora_bokstäver$("kalle") // OBS parametern EJ REF-ad!!
```

Detta skulle ge utskriften **KALLE**.

Följande minnesregel kan vara bra att ha i minnet när man använder funktioner:

Ett funktionsanrop kan förekomma *överallt* där en *variabel* kan ingå, *utom* där variabelns värde måste kunna ändras.

En funktion kan därför förekomma

- 1) I en utskriftssats
- 2) I beräkningsuttryck (t ex i högerledet av en tilldelningssats)
- 3) Som aktuell parameter till andra underprogram, där den formella parametern är *inparameter* (ej ”REF:ad”) – detta gäller t ex alla inbyggda funktioner i COMAL.

En funktion kan däremot *inte* förekomma

- 1) I vänsterledet till en tilldelningssats
- 2) Som "mottagare" av värde i en INPUT- eller READ-sats
- 3) Som aktuell parameter till ett underprogram där den formella parametern är *utparameter* ("REF:ad").

Avslutande kommentarer

Fördelarna med egendefinierade procedurer och funktioner är flera:

- De underlättar nedbrytning av problemställningar i delproblem, vilket avsevärt underlättar konstruktionen av stora och komplicerade program.
- En procedur/funktion kan anropas på flera ställen i ett program, vilket minskar programmets omfång.
- De gör programmen överskådligare, ty i procedurerna/funktionerna samlar man logiskt sammanhörande programdelar ("moduler"). Detta medför också att programmet blir lättare att förstå och underhålla, både för den som skrivit programmet och andra.
- Genom att samla nyttiga procedurer/funktioner i "bibliotek" kan programmeringsarbetet underlättas och effektiviseras.

Övningsuppgifter

4-1. Skriv om funktionen `siffersum#` (sid 244 i läroboken) utan att utnyttja VAL-funktionen (använd ORD-funktionen i stället).

4-2. Talen 36 och 24 har flera gemensamma divisorer (faktorer): 2, 3, 4, 6 och 12. Av dessa är talet 12 störst. Att bestämma största gemensamma divisorn till två heltal kan vara ganska besvärligt om talen är stora. Problemet kan emellertid lösas helt generellt med en gammal räkne-regel som kallas Euklides' algoritm.

Om de 2 talen betecknas `m#` och `n#` kan algoritmen skrivas så här i pseudokod:

```
Så länge som n# <> 0
  Sätt r# = resten av m#/n#
  Sätt m# = n#
  Sätt n# = r#
m# är den största gemensamma divisorn (faktorn)
```

Skriv en funktion, `sgd(m#, n#)`, som returnerar största gemensamma divisorn till `m#`, `n#`.

Testa funktionen med några talpar, t ex 105 och 234, 9876 och 5432.

4-3. Skriv en heltalsfunktion som tar emot en textsträng och bestämmer antalet ord i strängen. Räkna för enkelhets skull enbart blanktecken som ordavskiljare, dvs om t ex "Ring 121314 genast" skickas till funktionen ska 3 returneras. Funktionen ska klara av även extremfall, som att strängen är tom eller bara innehåller blanktecken (0 ord). *Tips*: Inför en boolsk variabel, `iord`, som håller reda på om man befinner sig i ett ord eller inte.

4-4. Skriv en funktion, `nybas$(decimal#, bas#)`, som omvandlar ett decimalt tal till en godtycklig talbas (2-36).

Kapitel 5

Rättelser

Sid 256:

Om man inte själv anger ett filtypstillägg lägger COMAL-systemet automatiskt till **.DAT** till filnamnet.

Sid 256 ff:

Funktionen `FILEFREE` heter **FREEFILE** i UniCOMAL.

Sid 261:

Byt ut kommatecknet på programrad 320 mot ett semikolon (för `ZONE`).

Sid 264:

Byt ut kommatecknen på programraderna 420 och 440 mot semikolon (för `ZONE`).

Sid 267:

Byt ut kommatecknen på programraderna 290 och 490 mot semikolon (för `ZONE`).

Ändra dessutom programraderna 480 och 530 till:

```
480     SELECT OUTPUT "PRN:"
530     SELECT OUTPUT "CON:"
```

Sid 268:

Proceduren `uppehåll` fungerar ej (se kommentarerna till sid 192 ovan). Den kan t ex formuleras så här i stället:

```
650 PROC uppehåll
660   PRINT AT 23,20: "Tryck valfri tangent för att fortsätta!"
670   WHILE KEY$ = "" DO NULL
680 ENDPROC uppehåll
```

Sid 268:

I uppgift 5.1a ska proceduren `skriv valuta` heta `skriv_valuta`.

Sid 268:

Hänvisningen i övning 5.2 gäller sidan 186 (ej sid 170).

Sid 269:

I övning 5.6 bör ”hittat” behandlas som en boolsk variabel (`TRUE/FALSE`) i stället för som ett tal.

Sid 273:

Övning 5.10: Använd TRUE/FALSE för att hålla reda på om medlemsavgiften är betald eller ej.

Övning 5.11: Använd en "boolsk" variabel, bokad.

Sid 275:

De 3 sista "COMAL-orden" skrivs på följande sätt i UniCOMAL:

```
SELECT "PRN:"  
SELECT "CON:"  
FREEFILE
```

Sid 294 (Facit):

Byt ut kommatecknen på raderna 1700, 1710 och 1730 mot semikolon.

Filhantering

Olika filtyper

Med avseende på lagringsformatet kan man skilja på två slag av filer: *binära filer* och *textfiler*. I en binär fil lagras data i samma ”kodade” format som används internt i datorn, medan data i en textfil utgör textrader, dvs är uppbyggd av ASCII-tecken, indelade i rader.

I läroboken behandlas bara (sekventiella) binärfiler. Här ska vi också ta upp textfiler och säga några ord om direktfiler.

I själva verket har du redan utnyttjat både binära filer och textfiler: När ett COMAL-program sparas med `SAVE` lagras det nämligen i binärformat och med `LIST/DISPLAY` lagras det i textformat.

Detta faktum ger dig en möjlighet att enkelt få en ”visuell” uppfattning om skillnaderna mellan de två filtyperna. Gör så här:

- 1) Spara ett COMAL-program både med `SAVE` och med `LIST` (el `DISPLAY`)
- 2) Lämna COMAL-systemet (`BYE`)
- 3) Titta på filerna med hjälp av DOS-kommandot **TYPE** (skriv `TYPE` följt av det fullständiga filnamnet)

Det program som sparats med `SAVE` kommer att åstadkomma en hel del underligheter på bildskärmen: mer eller mindre obegripliga tecken, pip m m omväxlar med läsbar text (om programmet innehöll text). Och så plötsligt kanske bara avbryts programlistningen innan filens slut nåtts.

Det program som sparades med `LIST` eller `DISPLAY` kommer däremot att se helt normalt ut: programrad efter programrad rullar fram på bildskärmen tills hela programmet listats (eller du själv avbrutit utskriften).

En textfil är alltså ”direkt” läsbar, medan en binärfil (normalt) inte är det.

En annan påtaglig skillnad mellan de två filslagen kan du studera genom att jämföra filstorleken av ett program sparats med `SAVE` och med `LIST`. Filstorleken erhåller du med `DIR`-kommandot. Som du själv kan konstatera är binärfilen mindre än motsvarande textfil, dvs data lagras (vanligen) kompaktare i binärfiler. Dessutom sparas/laddas program snabbare med `SAVE` och `LOAD` än med `LIST/DISPLAY` resp `ENTER`, vilket bl a beror på att COMAL-systemet i det senare fallet måste konvertera programmet från/till det binärformat som det har i datorns minne.

Det som här intresserar oss är inte i första hand programfiler, men det mesta som sagts ovan gäller också datafiler: Även datafiler kan vara av antingen binärt format eller textformat, med de kännetecken som beskrivits ovan. Vi ska nu titta lite närmare på hur de olika typerna av datafiler hanteras av COMAL-systemet.

Eftersom binära filer och textfiler lagras på olika sätt, använder COMAL olika satser för att skriva och läsa de olika filtyperna. Däremot öppnas och stängs (sekventiella) binär- och textfiler på samma sätt (med `OPEN FILE`- resp `CLOSE`-satserna), varför vi här inte specialbehandlar detta problem (OBS att i båda fallen erhåller filnamnet automatiskt filtypsbeteckningen **.DAT** om man inte anger något annat).

Sekventiella filer

Sekventiella filer kännetecknas av att filposterna måste behandlas i tur och ordning, från filens början och framåt till filens slut. Dessutom kan man enbart utföra en operation i taget på en sekventiell fil, dvs *antingen* läsa *eller* skriva på filen. Vill man *både* läsa *och* ändra filposter måste

man utnyttja en extra fil dit man skriver både de oförändrade och ändrade värdena – se läroboken.

Sekventiella binärfiler

Detta är den filtyp som behandlas i läroboken.

Läsning/skrivning sker med satserna:

```
WRITE FILE filnr#: <variabellista>
READ FILE filnr#: <variabellista>
```

Sekventiella textfiler

Läsning/skrivning sker med satserna:

```
PRINT FILE filnr#: <variabellista>
INPUT FILE filnr#: <variabellista>
```

Lägg märke till att man här använder varianter av `PRINT` och `INPUT`, vilket har en ”djupare” innebörd i det att bildskärmen (och tangentbordet) faktiskt kan ses ett specialfall av en textfil (det som visas på bildskärmen är ju vanligen *text*, inte det interna lagringsformatet av data).

Exempel

Med följande program kan vi skriva ut en textfil på bildskärmen:

```
DIM text$ OF 200, filnamn$ OF 13

INPUT "Ange filnamn: ": filnamn$
OPEN FILE 1,filnamn$,READ
WHILE NOT EOF(1) DO
  INPUT FILE 1: text$
  PRINT text$
ENDWHILE
CLOSE
PRINT
PRINT "KLART"
```

Testa gärna programmet på ett COMAL-program sparat i ASCII-format.

Utskrift på textfil fungerar i princip likadant som utskrift på bildskärm och skrivare (varför både `TAB` och `USING` kan användas, men däremot inte `PRINT AT`, som bara gäller bildskärmen). Detta betyder också att man måste tänka sig för om flera uttryck, särskilt numeriska sådana, ska skrivas på en fil. Om t ex flera tal, åtskilda med kommatecken, skrivs kommer de att ”packas” på samma sätt som på bildskärmen, varför man sedan inte kan komma åt de enskilda talen vid inläsning från filen. Man måste därför se till att uttrycken åtskiljs av minst ett mellanslag. Detta kan åstadkommas med semikolon (ger ett mellanslag om ej `ZONE` satts), med `TAB`-funktionen eller med `PRINT USING`. Alternativt kan man skriva ut varje uttryck med en egen `PRINT`-sats, dvs ett värde per rad.

Direktfiler

En direktfil kännetecknas av att de enskilda filposterna är åtkomliga i godtycklig ordning.

Filposterna i en direktfil har en fast längd, som måste specificeras när filen öppnas. Det är den fixerade postlängden som gör det möjligt att komma åt en godtycklig post: Om postens nummer, räknat från filens början, är känt, behöver COMAL-systemet i princip bara utföra en enkel multiplikation (postlängden*postnumret) för att lokalisera posten.

Det är vanligt (men ej nödvändigt) att skapa en direktfil av viss storlek innan man börjar lägga in data i den. För detta finns en speciell sats:

```
CREATE filnamn$,antal_poster#,postlängd#
```

där `postlängd#` anger en dataposts längd räknad i bytes (se nedan) och `antal_poster#` anger hur många poster som ska skapas i filen.

Om inget filtypstillägg anges, fogar COMAL-systemet automatiskt till **.RAN** i slutet av filnamnet.

Om man ska bearbeta en direktfil (för läsning/skrivning) öppnas den med satsen

```
OPEN FILE filnr#,filnamn$,RANDOM postlängd#
```

Postlängden för binära direktfiler kan beräknas utifrån följande

- 1) Strängar kräver 2 bytes + den dimensionerade stränglängden
- 2) Heltal kräver 4 bytes
- 3) Flyttal kräver 8 bytes

Exempel: En post med 2 strängar, dimensionerade till 80 tecken vardera, 1 heltal och 2 flyttal har postlängden: $2*(80+2)+4+2*8 = 184$.

Lägg märke till att en direktfil, i motsats till en sekventiell fil, öppnas på samma sätt oavsett om man ska läsa eller skriva på den, dvs när den väl är öppnad kan man både läsa och skriva på filen (men givetvis bara en sak i taget).

Skrivning/läsning på binära direktfiler görs, liksom i fallet med sekventiella binärfiler, med **WRITE FILE** och **READ FILE**, men förutom filnummer anges även numret på den post som ska överföras:

```
WRITE FILE filnr#,postnr#: <variabellista>
READ FILE filnr#,postnr#: <variabellista>
```

(Anm: Om `postnr#` inte anges skrivs/läses posten efter den som sist hanterades, dvs läsning/skrivning sker sekventiellt).

Skrivning/läsning på en direktfil av ASCII-format görs, på helt analogt sätt med

```
PRINT FILE filnr#,postnr#: <variabellista>
INPUT FILE filnr#,postnr#: <variabellista>
```

Direktfiler stängs på exakt samma sätt som sekventiella filer.

Övningsuppgifter

5-1. Skriv ett program som först skriver ut ett antal slumpstal på en binär (sekventiell) fil och sedan läser dessa från filen och skriver ut dem på bildskärmen.

Extrauppgift: Summera talen vid inläsningen och skriv ut antalet värden och deras totalsumma.

5-2. Lärarna på KOMVUX tycker det är jobbigt att räkna ut poängsumma och medelpoäng vid skrivningsrättning. Likaså är det jobbigt att räkna ut medelbetyg när terminsslutet närmar sig. Varför inte skriva ett program som utför detta? För att spara resultaten skriver vi dem på en fil (binärfil eller textfil). Uppgift:

- Skriv ett program som läser in resultaten från tangentbordet och skriver dem på en fil, vars namn användaren själv får bestämma. Inmatningen ska avslutas med att användaren skriver ut något speciellt värde, som givetvis inte skrivs på filen.
- Skriv ett program som läser in värdena från filen och (på skärmen) skriver ut antalet värden och deras medelvärde.

5-3. Har du prövat på en ordbehandlare? En ordbehandlare är ett datorprogram med vilket man inmatar och redigerar text. Ofta innehåller ordbehandlare en massa finesser som underlättar skrivarbetet. Du ska nu skriva en sådan ordbehandlare! Nej skämt åsido, det gäller att göra något mycket enklare: Ett program med vilket man å ena sidan kan skriva in och spara text på en fil, å andra sidan läsa text som finns på fil. Programmet ska vara menystyrt (avsluta, inskriva text, läsa text) och filnamnet ska kunna bestämmas av användaren. Textinmatningen avslutas lämpligen med att användaren skriver en tomrad. Läsning av text bör ske skärmvis så att inte texten bara rullar förbi.

5-4. Gör ett program som läser ett COMAL-program sparad med `DISPLAY` och skriver ut programmet på en ny textfil, nu med radnummer inlagda (glöm inte att radnumret måste följas av minst ett blanktecken). Pröva sedan att ladda och provköra den nya programfilen. Om du löst uppgiften korrekt ska det gå fint.

5-5. En mattelärare har ett tag dragit nytta av det program vi skapade i uppgift 5-2 och tycker det är ett utmärkt hjälpmedel. Nu har läraren fått mersmak på datorstöd i sitt arbete och klurat ut att det där med att skriva in skrivningsresultat i ”gröna fasan”, det borde ju kunna skötas bättre med en dator. Tyvärr är läraren inte programmeringskunnig och har vänt sig till oss för att få hjälp att ordna det hela. För att testa programmet har vi fått ett utdrag ur ”gröna fasan”. Det är fråga om namn och 4 skrivningsresultat för 20 elever:

Allan	3 3 2 2	Inga	2 1 1 2	Peter	4 3 3 3
Berta	4 5 4 4	Ivar	4 5 5 4	Rune	3 3 4 4
Dagmar	3 4 4 5	Lena	4 4 5 5	Siv	3 5 5 4
Elsa	3 3 5 4	Mikael	2 2 2 3	Stig	3 3 2 3
Erik	2 2 1 2	Nils	3 4 3 5	Tina	2 2 3 3
Gunnar	4 3 2 1	Nina	5 5 5 5	Ulf	1 2 1 1
Harald	3 3 3 3	Olof	3 3 3 3		

- Skriv ett program som skapar en fil ("`MAPROV.DAT`") med dessa data.
- Skriv ett program som läser filen och visar namnen på alla elever som haft betyget 5 på en viss skrivning (1-4) som bestäms av användaren.
- Skriv ett program som läser filen och skriver ut namn och medelbetyg för var och en av eleverna.

Följande uppgift är ganska omfattande och löses lämpligen i grupp.

5-6. Vi ska bygga upp ett register över en skivsamling. Registret ska ligga på en fil, som ska kunna uppdateras. Vi vill givetvis också kunna läsa och söka reda på skivor i registret. Allt ska kunna väljas från en meny.

Detta är en lite större uppgift, varför det är viktigt att dela upp den i delproblem. Om man börjar med att göra "huvudmenyn", eller gör delprogrammen först och menyn sist, är av mindre vikt, men vi väljer den senare metoden, ty därmed får vi tidigt fungerande delprogram, vilket kan vara bra om man inte är så van att jobba med större uppgifter.

På de följande sidorna ges detaljerade anvisningar om de olika delprogrammets uppbyggnad och funktion. Detta ger ganska liten frihet i utformningen, men det bör göra det möjligt att relativt snabbt slutföra uppgiften.

1) Inmatning

Det första som måste fastläggas är poststrukturen, dvs vilka fält (termer) som ska ingå och hur stora dessa ska vara. Vi väljer följande:

fältnr	fältnamn	antal tecken
1	skivnummer	2
2	skivnamn	25
3	artistnamn	30
4	musiktyp	10
5	skivtyp	8

Vi måste också bestämma skärmlayouten. Den bör se ut något i den här stilen:

```

SKIVREGISTRETINMATNING
=====
SKIVNUMMER2
SKIVNAMNVID HAVET
ARTISTNAMNULF LUNDELL
MUSIKTYPPOP/ROCK
SKIVTYPLP
=====

```

- Inmatningen ska pågå tills användaren skriver skivnumret 0.
- Eventuella meddelanden till användaren skrivs ut under ovanstående ruta.

Några tips:

- 1) Använd för enkelhets skull enbart strängar för de olika fälten.
- 2) Glöm inte att dimensionera.
- 3) Kalla skivfilen "SKIVREG.DAT".
- 4) Använd proceduren **teckenrad** för att skriva de streckade linjerna.
- 5) Låt programmet först fråga om det är ett nytt eller gammalt skivregister som ska hanteras. I det senare fallet öppnas filen med APPEND i stället för WRITE.
- 6) För "uppdateringen" på bildskärmen är det elegantast att behålla själva "skelettet" och bara radera det som inmatats. Detta är dock inte nödvändigt: Det går även bra att skriva om hela skärmen mellan inmatningen av varje post.
- 7) Tänk noga igenom hur inmatningen ska skötas (med WHILE- eller REPEAT-konstruktion).
- 8) Skriv ut en hel post i taget på filen.
- 9) Glöm inte att stänga filen.

2) Utskrift av skivregistret

På skärmen vill vi under utskriften (på skrivare) ha en liknande layout som vid inläsningen, men här räcker det med en rubrik:

```
SKIVREGISTERUTSKRIFT
=====
```

På papperet vill vi för det första ha en rubrik som talar om att det är fråga om ett skivregister och sedan ska det se ut något i den här stilen:

NR	SKIVNAMN	ARTISTNAMN	MUSIKTYP	SKIVTYP
1	FÖRBLINDAD	KAJSA ANKA	ROCK	EP
2	MODERNA TIDER	GYLLENE TIDER	POP	LP

Extrauppgifter:

- a) Gör det möjligt att välja mellan att få utskriften på skärm eller skrivare.
- b) Ordna så att programmet gör "sidbyte" (på skärm resp papper) när en sida är fullskriven. För detta är det nödvändigt att hålla räkning på antalet utskrivna rader. (OBS att "sidorna" på skärm resp papper är inte lika långa).

3) Söka skiva

För att göra det enkelt för oss tillåter vi bara sökning på skivnamn. Sökningen går till på så sätt att filen genomsöks från början till dess skivan hittats eller filslut nåtts (denna sökningsmetod, där elementen går igenom ett efter ett, kallas *linjär*). Om skivan finns ska all information om denna skrivas ut på skärmen, i annat fall upplyses vi om att den saknas i registret. Programmet ska därefter fråga om användaren vill söka fler skivor.

Skärmlayouten bör likna de övriga.

4) Uppdatera/ändra

Gör så här:

- Öppna skivfilen för läsning och en annan fil, "SKIVREG. \$\$\$", för skrivning
- Så länge som filen inte är slut:
 - Läs en post i taget från filen skriv ut den på bildskärmen.
 - Visa en meny: 1) Ändra 2) Ändra ej 3) Ta bort (Välj 1-3):
 - 1: Användaren får mata in den nya posten som sedan skrivs ut på utskriftsfilen
 - 2: Den gamla posten skrivs ut på nya filen
 - 3: Ingenting görs åt posten, i stället inläses nästa post från läsfilen
- Stäng båda filerna
- Radera första filen och omdöp den nya till SKIVREG.DAT

För övrigt: Försök få skärmlayouten att likna de övriga. Ett sätt att göra det är att använda samma skärmutseende som vid inmatningen och låta vår lilla meny stå under det nedre strecket.

5) Huvudmenyn

När ovanstående deluppgifter avklarats är det dags att sammanlänka det hela. Detta kan göras på olika sätt. Man kan t ex föra ihop allt i ett enda program (varvid delprogrammen görs om till procedurer). Det går också att låta delprogrammen förbli självständiga program och anropa dem från huvudmenyn som då också blir ett självständigt program (det görs med satsen **CHAIN** – se nedan). För övnings skull gör vi här på det andra sättet.

Och så till sist:

Dela upp programmet/programmen i lätthanterliga procedurer/funktioner. Principen bör vara att varje procedur/funktion ska göra *en* sak, men bra!

Utnyttja om möjligt programsnuttar som du gjort tidigare (ibland kanske de går använda direkt, i andra fall krävs mindre justeringar).

5-7. En nackdel med att hantera poster som i föregående övning är att samtliga fältnamn måste anges när en post ska skrivas/läsas, eller om en post ska överföras som parameter från/till ett underprogram. Önskvärt är att man i sådana fall skulle kunna hantera en post som en enda enhet. I vissa programspråk (som Pascal) finns speciella datatyper (records) just för detta ändamål. Så inte i vår COMAL-version. Det finns dock ett sätt att "simulera" en sådan posttyp: Vi låter posten representeras av en indexerad variabel, där fälten utgörs av den indexerade variabelns element:

```
skivpost$(1)      skivnummer
skivpost$(2)      skivnamn
skivpost$(3)      artistnamn
etc
```

Med tilldelningarna:

```
skivnr#:=1
skivnamn#:=2
etc
```

kan sedan ”fältnamnen” (i form av heltalsvariabler) användas i stället för de anonyma indexen när vi vill komma åt enskilda fält (termer):

```
skivpost$(skivnr#)
skivpost$(skivnamn#)
etc
```

Med dessa konstruktioner vinner vi bl a följande:

- 1) Hela posten kan skrivas/läsas på en gång genom att bara ange namnet på den indexerade variabeln, `tex READ FILE filnr#: skivpost$()`
- 2) Hela posten kan överföras till/från underprogram som en enda parameter (`skivpost$()`)

Finns då inga nackdelar? Jomennisst!

- 1) Den indexerade variabeln måste dimensioneras efter den längsta i posten ingående strängen (i vårt exempel `DIM skivpost$(5) OF 30`), vilket är minnesslöseri. När det gäller hanteringen av poster i datorns primärminne är detta problem lyckligtvis ganska marginellt, eftersom vi nästan alltid bara behandlar en eller högst några få poster åt gången. Däremot kan det få betydelse för skivutrymmet, ty där kan ett stort antal poster finnas.
- 2) I en indexerad variabel måste samtliga element ha samma datatyp. Om bara hel- och flyttal ingår i posten kan vi låta elementen i den indexerade variabeln utgöras av flyttal, men om strängar ingår måste allafälten representeras som strängar, även de som innehåller antal, pris osv. Förutom att strängar kräver mer minnes än tal, måste vi, när beräkningar ska göras på sådana fält, först konvertera dem till tal (med `VAL`-funktionen) och sedan konvertera tillbaka till strängar igen (med `STR$`-funktionen).

Trots dessa nackdelar överväger ofta fördelarna. För att få en inblick i tekniken tillämpar vi den på vårt skivregister. Eftersom samtliga fält där redan är strängar, slipper vi besväret att konvertera från och till tal, vilket underlättar en hel del. Programmen behöver givetvis inte skrivas om helt: Gör nödvändiga förändringar i de befintliga programmen. Observera dock att den gamla datafilen inte kan användas med det nya programmet.

CHAIN-satsen

`CHAIN` används för att avsluta körningen av ett program och automatiskt ladda och köra (”exekvera”) ett annat program. Syntax:

```
CHAIN filnamn$
```

Det program som ska ”chainas” måste vara lagrat med `SAVE`-kommandot.

Appendix 1 – För den som vill veta mer

Det som behandlas här är överkurs, men kan komma till användning i t ex projektarbeten.

Mera om **SELECT OUTPUT**

Tidigare har vi utnyttjat **SELECT OUTPUT**-satsen för att omdirigera utmatning till skrivare. Satsen kan även användas för att omdirigera utmatning till en textfil på skiva. Syntax:

```
SELECT OUTPUT filnamn$
```

Om inget filtypstillägg anges, lägger COMAL-systemet automatiskt till **.DAT**, precis som för vanliga (sekventiella) datafiler.

Möjligheten att omdirigera utmatning till en fil kan vara mycket användbar (man kan t ex ta in filen i en ordbehandlare).

Tidmätning

I många sammanhang behöver man kunna mäta tidsåtgång. Det kan gälla allt ifrån spelprogram till styr/mät-tillämpningar. I UniCOMAL finns en sats, **TIMER**, med vilken sådana tidmätningar kan göras. **TIMER** kan både användas för att avläsa den inbyggda "klockan" och för att sätta den till ett bestämt värde:

```
t :=TIMER
```

returnerar tiden i sekunder (med 2 decimaler, dvs 100-delar), medan

```
TIMER t
```

sätter klockan till tiden t (vanligt är att först nollställa tiden när man ska göra en tidmätning, dvs låta t få värdet 0).

Observera att **TIMER** är till för tidmätning, inte för att avläsa systemklockans tid (tim:min:sek). Det senare görs med en annan funktion, strängfunktionen **TIME\$**.

Sortering

Sorteringsproblemet är mycket viktigt och det finns ett otal sorteringsmetoder utvecklade för olika användningsområden: för att sortera stora och små datamängder, för att sortera i datorns primärminne (ofta ligger då data i indexerade variabler) och för att sortera på yttre minnen (filsortering).

I läroboken har du träffat på en enkel sorteringsmetod för data i indexerade variabler: Urvals-sortering (övningsuppgift 4.48). Denna tillhör inte de effektivaste metoderna, men den duger bra för små datamängder och för att introducera sorteringsproblemet. Vi ska här utan kommentarer redovisa en betydligt effektivare metod: *Shellsortering*. Liksom Urvalssortering kan Shell-sortering användas för att sortera data i indexerade variabler och är lämplig då man har att göra med medelstora datamängder. För ännu större datamängder eller om ännu större effektivitet är önskvärd brukar man tillgripa andra metoder (såsom *Quicksortering*).

```

// Sorterar heltal. För andra datatyper måste den indexerade
// variabelns typ ändras, liksom variablerna i byt-proceduren
PROC shellsort(antal#,REF tal#()) CLOSED
  steg#:=antal#
  WHILE steg#>1 DO
    steg#:=steg# DIV 2; m#:=antal#-steg#
    REPEAT
      ej_bytt:=TRUE
      FOR v#:=1 TO m# DO
        h#:=v#+steg#
        IF tal#(v#)>tal#(h#) THEN
          byt(tal#(v#),tal#(h#))
          ej_bytt:=FALSE
        ENDIF
      ENDFOR v#
    UNTIL ej_bytt
  ENDWHILE

  PROC byt(REF tal1#,REF tal2#) CLOSED
    temp#:=tal1#; tal1#:=tal2#; tal2#:=temp#
  ENDPROC byt
ENDPROC shellsort

```

Övningsuppgift

A-1. Skriv ett program som testar `shellsort`-proceduren ovan. Mät sorteringstiden med `TIMER`. Jämför gärna med motsvarande tider för Urvalssortering.

Binärsökning

Vi har redan träffat på en sökningsmetod som kallas för *linjär*. Linjärsökning kan användas både med indexerade variabler och filer (se sakregistret och övningsuppgift 5.6 i läroboken). Linjärsökning går till på så sätt att man startar i början av den indexerade variabeln eller filen och undersöker elementen ett efter ett tills dess det sökta hittats eller alla element kontrollerats.

En fördel med metoden är att det inte spelar någon roll hur data är organiserade. Om data är *sorterade* finns dock betydligt effektivare sökningsmetoder. En sådan är *binärsökning*. Även denna metod kan användas för både indexerade variabler och filer (direktfiler).

Binärsökning går ut på att ”inringa” det sökta elementet genom att först undersöka i vilken halva (binär betyder två) av den sorterade datamängden där det sökta elementet eventuellt finns och sedan fortsätta sökningen i denna halva på samma sätt. Detta görs genom att det sökta elementet hela tiden jämförs med mittenelementet i den del av datamängden som undersöks. Till slut har vi antingen funnit det vi söker eller genomsökt alla data.

I ”pseudokod” kan metoden beskrivas så här:

```

Upprepa
  mitten:= (nedre gräns + övre gräns) DIV 2
  Om mittenelementet = det sökta
    så är det klart
  annars om det sökta elementet > mittenelementet så
    fortsätt sökningen i övre halvan, dvs nedre gräns:=mitten+1
  annars
    fortsätt sökningen i nedre halvan, dvs övre gräns:=mitten-1
  tills elementet hittats eller hela ”listan” genomsökts

```

Hur ”algoritmen” kodas beror delvis på tillämpning. Vi ger här ett exempel på hur detta kan göras för en indexerad variabel med heltal som element. Det hela utförs i en procedur, med det sökta elementet, den indexerade variabeln och sökningens undre och övre indexgränser som inparametrar (den indexerade variabeln har ”reffats” för att spara minne och tid). Utparametrar är det index som proceduren kommit fram till, samt en ”boolsk variabel” som anger om elementet hittats eller inte:

```
PROC binsök(sökt#,REF a#(),undre#,övre#,REF mitt#,REF funnen#)
  funnen#:=FALSE
  REPEAT
    mitt#:=(undre#+övre#) DIV 2
    IF a#(mitt#)=sökt# THEN
      funnen#:=TRUE
    ELIF sökt#>a#(mitt#) THEN
      undre#:=mitt#+1
    ELSE
      övre#:=mitt#-1
    ENDIF
  UNTIL funnen# OR undre#>övre#
ENDPROC binsök
```

Exempel på användning, med den *sorterade* indexerade variabeln `tal#()` och dess indexgränser (1 resp `antal#`) givna:

```
INPUT "Ange sökt element: ": sökt#
binsök(sökt#, tal#(), 1, antal#, index#, funnen#)
IF funnen# THEN
  PRINT "Talet fanns! Det hade index";index#
ELSE
  PRINT "Talet fanns inte"
ENDIF
```

För att testa det hela kan förslagsvis ovanstående kombineras med lösningen till övningsuppgiften i avsnittet om Shellsortering.

Felhantering

Många gånger har du säkert råkat ut för programkrascher av olika anledningar: Du skrev en bokstav vid inmatning till en talvariabel, en fil som skulle öppnas för läsning fanns ej, nämnaren var noll vid en division osv. Ett användarvänligt program får givetvis inte stoppa på sådana fel. I stället bör programmet ge användaren lämpliga felmeddelanden och möjligheter att korrigera felen.

COMAL innehåller en kraftfull konstruktion för att felövervaka och ta hand om fel. Konstruktionen, som inspirerats av programspråket Ada, ser ut så här:

```
TRAP
  <satser som ska felövervakas>
HANDLER
  <satser som utförs om fel uppstår>
ENDTRAP
```

De satser som skall felövervakas (”TRAPPas”) placeras mellan nyckelorden `TRAP` och `HANDLER`. Om inget fel inträffar kommer de att utföras på vanligt sätt. Vid fel avbryts exekveringen av dessa satser och programkörningen fortsätter med satserna efter `HANDLER`.

Några exempel får illustrera det hela.

1. Undersök om en fil finns eller ej.

```
FUNC fil_finns(filnamn$) CLOSED
  filnr#:=FREEFILE
  TRAP
    OPEN FILE filnr#,filnamn$,READ
    finns#:=TRUE
  HANDLER
    finns#:=FALSE
  ENDTRAP
  CLOSE FILE filnr#
  RETURN finns#
ENDFUNC fil_finns
```

2. Kontrollera numerisk inmatning. Den inmatade strängen skickas till funktionen nedan. Om strängen kan omvandlas till ett tal returneras TRUE, annars FALSE:

```
FUNC numeriskt(sträng$) CLOSED
  ok#:=TRUE
  TRAP
    y:=VAL(sträng$)
  HANDLER
    ok#:=FALSE
  ENDTRAP
  RETURN ok#
ENDFUNC numeriskt
```

3. I kombination med **RETRY**-satsen kan man åstadkomma att satserna mellan TRAP och HANDLER upprepas tills inmatningen är felfri:

```
TRAP
  INPUT "Inmata tre tal:": första,andra,tredje
HANDLER
  RETRY
ENDTRAP
```

Den som närmare vill studera felhanteringsmöjligheterna med TRAP-HANDLER kan titta i handboken *COMAL COMAL COMAL*, som bör finnas tillgänglig i lektionssalen (fråga din lärare). En översikt ges på sidorna 297-302. Se även uppslagsorden ENDTRAP, ERR, ERRFILE, ERRTEXT\$, HANDLER, TRAP och REPORT. Handboken är skriven för VIC-COMAL, men det mesta fungerar likadant med vår COMAL (lägg dock märke till att andra regler gäller för filnamn).

Färger m m

Om du jobbar vid en färgskärm kan färgen på bakgrund och text varieras. Med s k monokroma skärmar är möjligheterna mer begränsade, men det går att erhålla utskrift i omvänd video (svart text på ljus bakgrund), liksom understruken, blinkande och extra fet (ljus) text.

Exempel på färganvändning finner du i COMAL-lektion nr 10. Skulle du behöva mer information, hittar du sådan i UniCOMAL-manualen.

”Monokroma” effekter kan erhållas på ett enklare sätt med att bifoga s k styrtecken till PRINT-satser. Vilka styrtecken som ger vilka effekter framgår av manualen, appendix G, sid 3 (de styrtecknen det gäller är CHR\$(16)–CHR\$(25)).

Grafik

UniCOMAL ger tillgång till ett kraftfullt ”grafikpaket”. Med detta kan man att rita upp matematiska funktioner, simulera fysikaliska fenomen, åstadkomma animerade bilder, ”skapa konst” m m.

Lärobokens grafikkapitel är anpassat till Compis-datorn och kan inte direkt tillämpas på våra maskiner, ty det finns en hel del skillnader mellan grafikinstruktioner och ”hårdvara”. Alla programexemplen finns dock omskrivna så att de är körbara med vår COMAL – se kapitel 2 i detta kompendium.

En introduktion till ”grafikpaketet” ges i de grafiklektioner som ingår i COMAL-lektionerna. En fullständig redogörelse för grafikmöjligheterna finns i COMAL-manualen. Exempelprogram med grafik finns på en specialdiskett som medföljer COMAL-systemet (kontrollera med din lärare).

Hexadecimala och binära tal

I en del tillämpningar är det naturligt att arbeta med binära och/eller hexadecimala. I UniCOMAL kan hexadecimala och binära tal användas överallt där *heltal* är tillåtna.

Hexadecimala tal skall föregås av ett *dollartecken* (\$):

```
hex# := $FF
PRINT hex# // Ger 255
```

De hexadecimala siffrorna A-F kan skrivas med små eller stora bokstäver (COMAL-systemet omvandlar dem till stora).

Binära tal anges på motsvarande sätt med ett *procenttecken* (%) före själva talet:

```
bin# := %1000
PRINT bin# // Ger 8
```

Det är även möjligt att inmata hextal (och binära tal) med INPUT-satser:

```
INPUT "Skriv ett hextal (inlett av ett $-tecken):": hex#
PRINT hex#
```

Hexadecimala och binära tal kan blandas i uttryck:

```
summa# := %1010 + $AB
PRINT summa# // Ger 181
```

VAL-funktionen tillåter hexadecimala eller binära talsträngar:

```
PRINT VAL("$f0") // Ger 240
PRINT VAL("%111") // Ger 7
```


Appendix 2

Systemkommandon

AUTO	Ger automatisk radnumrering vid inskrivning av program. Kommandot skrivs AUTO <första radnumret> <,steglängd> Anges ej första radnumret, sätts detta till sista radnumret + steglängden om programrader redan finns och 10 annars. Anges ej steglängd sätts denna till 10. AUTO 100 ger första radnumret = 100 och steglängden 10. AUTO 100,5 ger första radnumret = 100 och steglängden 5. Redan existerande radnummer markeras med omvänd video. Radnumreringen stoppas med Ctrl-C eller Ctrl-Break .
BYE	Används för att lämna COMAL-systemet, dvs återgå till operativsystemet (DOS).
CAT	Listar namnen på alla filer och program som finns på skiva (samma som DIR). CAT (aktuell skivenhet) CAT "A:" (skivenhet A) CAT "B:" (skivenhet B)
CON	Återupptar exekveringen av ett program som stoppats med Ctrl-Break eller STOP -satsen.
DEL	Raderar en eller flera programrader. DEL <område> DEL 10 raderar rad 10 DEL 10- raderar samtliga rader f o m rad 10 DEL 10-90 raderar f o m rad 10 t o m rad 90 DEL -70 raderar samtliga rader t o m rad 70 DEL namn raderar namngiven procedur eller funktion
DELETE	Används för att radera program eller filer från skiva. DELETE "<enhet:>prognamn" Om enhet utelämnas sätts den till aktuell skivenhet. DELETE "prognamn" (aktuell skivenhet) DELETE "A:prognamn" (skivenhet A) DELETE "B:prognamn" (skivenhet B)
DIR	Samma som CAT (se ovan).
DISPLAY	Ger listning av ett program på skärm, skrivare eller annan utenhet. Listningen sker utan radnummer. Syntax: se LIST -kommandot nedan.
EDIT	Används för att lista ett program radvis på skärmen när man vill ändra i programmet. Listningen sker inte kontinuerligt, såsom vid LIST -kommandot. Listningen avbryts med Ctrl-C el Ctrl-Break . EDIT <område> EDIT listar samtliga rader EDIT 10 listar rad 10 EDIT 10- listar samtliga rader f o m rad 10 EDIT 10-90 listar f o m rad 10 t o m rad 90 EDIT -70 listar samtliga rader t o m rad 70 EDIT namn listar procedur eller funktion

- ENTER** Används då program sparade med **LIST**- eller **DISPLAY**-kommandot skall läsas in från skivenhet.
ENTER "<enhet:> programn"
 Om enhet utelämnas används aktuell skivenhet.
ENTER "programn" (aktuell skivenhet)
ENTER "A:programn" (skivenhet A)
ENTER "B:programn" (skivenhet B)
- LIST** Ger utskrift av ("listar") programmet på skärm, skrivare eller annan utenhet. Listningen sker med radnummer.
LIST <område>
LIST listar samtliga rader
LIST 10 listar rad 10
LIST 10- listar samtliga rader f o m rad 10
LIST 10-90 listar f o m rad 10 t o m rad 90
LIST -70 listar samtliga rader t o m rad 70
LIST namn listar namngiven procedur eller funktion
 Listningen kan stannas upp och återstartas med tryck på mellanslagstangenten. Listningen avbryts med *Ctrl-Break*.
LIST-kommandot används även för att spara program som textfil (ASCII-fil) på yttre enhet:
LIST <område <TO> "<enhet:> programn"
 Om **enhet** utelämnas utnyttjas aktuell skivenhet.
 Om **område** utelämnas listas hela programmet.
LIST "programn" (aktuell skivenhet)
LIST "A:programn" (skivenhet A)
LIST "B:programn" (skivenhet B)
LIST "PRN:" (skrivare)
- LOAD** Används för att läsa in ett program sparat med **SAVE**-kommandot från yttre enhet (flexskiva el hårddisk).
LOAD "<enhet:>programn"
 Om enhet utelämnas utnyttjas aktuell skivenhet.
LOAD "programn" (aktuell skivenhet)
LOAD "A:programn" (skivenhet A)
LOAD "B:programn" (skivenhet B)
- NEW** Raderar programminnet.
- PASS** Används för att skicka kommandon till DOS.
- RENUM** Används för att omnumrera ett program. Kommandot skrivs
RENUM <område;> <startnr> <,steg>
område = procedurnamn, funktionsnamn eller radnummer-område (<startradnr> <-> <slutradnr>).
startnr = första nya radnummer
steg = steglängd
 Anges ej område, omnumreras samtliga rader.
 Anges ej första radnummer blir detta 10.
 Anges ej steglängd blir denna 10.
RENUM ger första radnumret = 10 och steglängden 10.
RENUM 100 ger första radnumret = 100 och steglängden 10.
RENUM 100,5 ger första radnumret = 100 och steglängden 5.
RENUM 100-;1000 ger första radnumret = 1000 och steglängden 10, men endast programraderna f o m 100 och uppåt omnumreras.

- RUN** Exekverar (kör) ett program.
RUN exekverar det program som finns i minnet.
RUN "<enhet:> programn" läser in och exekverar program sparad med **SAVE**-kommandot på yttre enhet.
- SAVE** Används för att spara ett program på yttre enhet (flexskiva el hårddisk).
SAVE "<enhet:>programn"
 Om enhetsbeteckning utelämnas utnyttjas aktuell skivenhet.
SAVE "programn" (aktuell skivenhet)
SAVE "A:programn" (skivenhet A)
SAVE "B:programn" (skivenhet B)
- SELECT** Används för att välja utenhet vid utskrift. Kommandot lyder **SELECT OUTPUT <typ>**, men **OUTPUT** kan utelämnas. Utmatning kan även styras till en textfil.
SELECT "PRN:" skrivare utenhet
SELECT "CON:" skärm utenhet
SELECT "MINFIL" utmatning till textfilen MINFIL
- SIZE** Visar hur många bytes av primärminnet som upptas av program respektive data, samt hur mycket ledigt minne som återstår.
- UNIT** Används för att ändra eller ta reda på aktuell skivenhet.
UNIT "<enhet>"
UNIT "A:" (skivenhet A)
UNIT "B:" (skivenhet B)
 Vilken skivenhet som är aktuell kan kontrolleras med **PRINT UNIT**

Editering m m

Tangent	Funktion
<i>Back Space</i>	Flyttar texten till höger om markören åt vänster och raderar framför sig.
<i>Enter</i> (↵)	Den inskrivna informationen sänds till COMAL-systemet, där den behandlas och eventuellt registreras.
<i>Skift</i>	Med skifftangenten uppe erhålls små bokstäver (gemener) och med tangenten nere erhålls stora bokstäver (versaler).
<i>Caps Lock</i>	Skiftar från små till stora bokstäver och tvärtom.
↑	Flyttar markören uppåt.
↓	Flyttar markören nedåt.
Fel! Hittar inte referenskälla.	
←	Flyttar markören åt vänster.
<i>Del</i>	Raderar tecknet under markören och flyttar resten av raden ett steg åt vänster.
<i>End</i>	Flyttar markören till slutet av aktuell rad.
<i>Home</i>	Flyttar markören till början av aktuell rad.
<i>Ins</i>	Skiftar mellan infognings- och överskrivningsläge.
<i>PgDn</i>	Rullar programmet på bildskärmen nedåt.
<i>PgUp</i>	Rullar programmet på bildskärmen uppåt.
<i>Tab</i>	Flyttar markören åt höger till nästa tabulatorstopp.
<i>Skift-PrtSc</i>	Kopierar bildskärmens innehåll till skrivare.
<i>Ctrl-Break</i>	Avbryter editering av en programrad. Avbryter programlistning. Stänger av AUTO -funktionen. Stoppas programexekvering.
Ctrl-Fel! Hittar inte referenskälla.	
<i>Ctrl-←</i>	Flyttar markören ett ord bakåt.
<i>Ctrl-End</i>	Raderar från markören till radens slut.
<i>Ctrl-Home</i>	Rensar bildskärmen.
<i>Ctrl-PgDn</i>	Flyttar markören till programmets sista rad.
<i>Ctrl-PgUp</i>	Flyttar markören till programmets första rad.
<i>Ctrl-C</i>	Avbryter redigeringen av en rad. Stänger av AUTO -funktionen.
<i>Ctrl-G</i>	Ger pip.

<i>Ctrl-M</i>	Motsvarar ↵.
<i>Ctrl-P</i>	Förhöjd ljusstyrka PÅ .
<i>Ctrl-Q</i>	Förhöjd ljusstyrka AV .
<i>Ctrl-R</i>	Omvänd video PÅ .
<i>Ctrl-S</i>	Omvänd video AV .
<i>Ctrl-T</i>	Blinkande text PÅ .
<i>Ctrl-U</i>	Blinkande text AV .
<i>Ctrl-V</i>	Understrykning PÅ .
<i>Ctrl-W</i>	Understrykning AV .
<i>Ctrl-X</i>	Ingen utskrift på skärmen.
<i>Ctrl-Y</i>	Återställer normal bildskärm.
<i>Alt-A</i>	Återställer ändrad programrad till ursprungligt utseende.
<i>Alt-B</i>	Markerar nedre högra hörnet av textfönster.
<i>Alt-D</i>	Raderar raden där markören står.
<i>Alt-E</i>	Raderar från markören åt höger (resten av skärmen).
<i>Alt-F</i>	Sätter textfönster till hela bildskärmen.
<i>Alt-I</i>	Infogar ny rad på den rad markören befinner sig.
<i>Alt-N</i>	Infogar ny programrad efter aktuell rad.
<i>Alt-O</i>	Återställer skärmen till standardläge och raderar bildskärmen.
<i>Alt-R</i>	Raderar programrad (även från minnet).
<i>Alt-T</i>	Markerar övre vänstra hörnet av textfönster.

Funktionstangenter

<i>F1</i>	LIST
<i>F2</i>	RUN + ↵
<i>F3</i>	AUTO
<i>F4</i>	RENUM
<i>F5</i>	DIR + ↵
<i>F6</i>	EDIT
<i>F7</i>	CON + ↵
<i>F8</i>	LOAD ”
<i>F9</i>	FIND ”
<i>F10</i>	SELECT OUTPUT ”CON:”
<i>Skift-F1</i>	LIST + ↵
<i>Skift-F2</i>	SCAN + ↵
<i>Skift-F3</i>	DEL
<i>Skift-F4</i>	SIZE + ↵
<i>Skift-F5</i>	CHDIR ”
<i>Skift-F6</i>	EDIT + ↵
<i>Skift-F7</i>	ENTER ”
<i>Skift-F8</i>	SAVE ”
<i>Skift-F9</i>	CHANGE ”
<i>Skift-F10</i>	SELECT OUTPUT ”PRN:”
<i>Ctrl-F1</i>	LIST ”
<i>Ctrl-F2</i>	RUN (automatiskt från filkatalog)
<i>Ctrl-F3</i>	MERGE (automatiskt från filkatalog)
<i>Ctrl-F4</i>	MERGE ”
<i>Ctrl-F5</i>	MKDIR ”
<i>Ctrl-F6</i>	DELETE (automatiskt från filkatalog)
<i>Ctrl-F7</i>	ENTER (automatiskt från filkatalog)
<i>Ctrl-F8</i>	LOAD (automatiskt från filkatalog)
<i>Ctrl-F9</i>	DELETE ”
<i>Ctrl-F10</i>	SELECT INPUT ”

Lösningar

De lösningar som här redovisas gäller enbart uppgifter i kompendiet. Radnummer har ej medtagits i programlistningarna.

```
// Övning 1.13
```

```
a)
```

```
PRINT "Moms på 6800 kr =" ; 25/100*6800 ; "kr"
```

```
Moms på 6800 kr = 1700 kr
```

```
b)
```

```
PRINT "Pris inkl. moms =" ; 1.25*2400 ; "kr"
```

```
Pris inkl. moms = 3000 kr
```

```
// Övning 1.17
```

```
PAGE
```

```
PRINT "En dator består av 4 huvuddelar"
```

```
PRINT
```

```
PRINT "1) Inenhet (tangentbord)"
```

```
PRINT "2) Centralenhet (processenhet och primärminne)"
```

```
PRINT "3) Yttre minne (sekundärminne)"
```

```
PRINT "4) Utenhet (bildskärm eller skrivare)"
```

```
// Övning 1.20
```

```
PAGE
```

```
PRINT AT 10,30: "*****"
```

```
PRINT AT 11,30: "***                ***"
```

```
PRINT AT 12,30: "***  Mina memoarer  ***"
```

```
PRINT AT 13,30: "***                ***"
```

```
PRINT AT 14,30: "*****"
```

```
// Övning 1.35
```

```
DIM förrätt$ OF 30, varmrätt$ OF 30, efterrätt$ OF 30, datum$ OF 12
```

```
förrätt$:="Vårrulle"
```

```
varmrätt$:="Friterade räkor med sötsur sås"
```

```
efterrätt$:="Friterade bananer med glass"
```

```
datum$:="15 mars 1988"
```

```
PAGE
```

```
PRINT AT 6,25: "Matsedel den";datum$
```

```
PRINT AT 7,25: "=====
```

```
PRINT AT 9,25: förrätt$
```

```
PRINT AT 11,25: varmrätt$
```

```
PRINT AT 13,25: efterrätt$
```

```
// Övning 1.40
DIM datum$ OF 8, namn$ OF 20, gadr$ OF 20, padr$ OF 20
DIM dag$ OF 7, tid$ OF 5, undertecknare$ OF 20
```

```
INPUT "Datum (ÅÅMMDD): ": datum$
INPUT "Namn: ": namn$
INPUT "Gatuadress: ": gadr$
INPUT "Postadress: ": padr$
INPUT "Vilken veckodag? ": dag$
INPUT "Vilken tid? ": tid$
INPUT "Vem skriver brevet? ": undertecknare$
PAGE
PRINT TAB(20);datum$
PRINT namn$
PRINT gadr$
PRINT padr$
PRINT
PRINT "Hej!"
PRINT "Vi träffas i klubbstugan på";dag$
PRINT "kl";tid$;"för taktikgenomgång inför matchen."
PRINT undertecknare$
```

```
// Övning 1.41
```

- a) 6
- b) 1
- c) 11.24
- d) 0

```
// Övning 1.45
```

```
PAGE
PRINT "Omvandling av temperatur, Celsius -> Fahrenheit"
INPUT "Ange gradtal i Celsius: ": tempc
tempf:=9*tempc/5+32
PRINT "Temperatur i Fahrenheit =";tempf
```

```
// Övning 1.49
```

```
// Banken
saldo:=5000
INPUT "Ange uttag/insättning med - eller + framför beloppet: ": trans
saldo:=saldo+trans
PRINT "Saldo:";saldo
```

```
// Övning 1.52
```

```
// Variabelnamnet ord$ är otillåtet. Det är ett reserverat COMAL-ord
DIM o$ OF 6
INPUT AT 2,1,3: "Ange ordet (högst 3 bokstäver): ": o$
o$:=o$(1:1)+o$(1:1)+o$(2:2)+o$(2:2)+o$(3:3)+o$(3:3)
PRINT o$
```

```
// Övning 1.53
```

```
DIM t$ OF 50,förk$ OF 5
t$:="American Standard Code for Information Interchange"
förk$:=t$(1:1)+t$(10:10)+t$(19:19)+t$(28:28)+t$(40:40)
PRINT förk$
```

```
// Övning 1.54
DIM fnamn$ OF 15, enamn$ OF 15
```

```
PAGE
INPUT AT 2,1,15: "Ange förnamn (högst 15 tkn): ": fnamn$
INPUT AT 3,1,15: "Ange efternamn (högst 15 tkn): ": enamn$
// a)
PRINT enamn$;fnamn$(1:1)
// b)
PRINT fnamn$;enamn$(1:1), "."
// c)
PRINT fnamn$(1:1), enamn$(1:2)
```

```
// Övning 3-1
DIM sträng1$ OF 80, sträng2$ OF 80
```

```
//Inmatning
INPUT "Ange sträng 1: ": sträng1$
INPUT "Ange sträng 2: ": sträng2$
//Bearbetning, jämförelse, utskrift
IF sträng1$<sträng2$ THEN
  PRINT sträng1$;"<";sträng2$
ELIF sträng2$<sträng1$ THEN
  PRINT sträng2$;"<";sträng1$
ELSE
  PRINT sträng1$;"=";sträng2$
ENDIF
```

```
// Övning 3-4
INPUT "Skriv ett tal mellan 0 och 1: ": tal
korrekt:=tal>0 AND tal<1
IF korrekt THEN
  PRINT "Korrekt"
ELSE
  PRINT "Felaktigt tal"
ENDIF
```

3-5 a) 0 b) 1 c) 4 d) 0 e) 48 f) 1 g) 5 h) 5

3-6 a) 0.6 b) 1 c) 0 d) 2 e) 8 f) 50

```
// Övning 3-7
INPUT "Skriv årtal: ": år#
skottår:=(år# MOD 4=0 AND NOT år# MOD 100=0) OR år# MOD 400=0
IF skottår THEN
  PRINT "skottår"
ELSE
  PRINT "ej skottår"
ENDIF
```

```

// Övning 3-8
PRINT "DIVISION MED ÖNSKAD NOGGRANNHET"
PRINT
INPUT "Ange täljaren (heltal): ": täljare#
INPUT "Ange nämnaren (heltal): ": nämnare#
INPUT "Ange antal decimaler: ": decant#
PRINT
heltalsdel#:=täljare# DIV nämnare#
PRINT täljare#,"/",nämnare#;"=";heltalsdel#,"",",
FOR i#:=1 TO decant# DO
  rest#:=täljare# MOD nämnare#
  täljare#:=rest#*10
  PRINT täljare# DIV nämnare#,
ENDFOR i#

// Övning 4-1
FUNC siffersum#(s$) CLOSED
  DIM tkn$ OF 1

  sum#:=0
  FOR i#:=1 TO LEN(s$) DO
    tkn$:=s$(i#:i#)
    IF tkn$>="0" AND tkn$<="9" THEN
      sum#:=sum#+ORD(tkn$)-ORD("0")
    ENDIF
  ENDFOR i#
  RETURN sum#
ENDFUNC siffersum#

// Övning 4-2
FUNC sgd(m#,n#) CLOSED
  WHILE n#<>0 DO
    r#:=m# MOD n#
    m#:=n#
    n#:=r#
  ENDWHILE
  RETURN m#
ENDFUNC sgd

// Testprogram
INPUT "Skriv två tal: ": m#,n#
PRINT "Största gemensamma divisorn är:";sgd(m#,n#)

```

```

// Övning 4-3
FUNC ordantal$(s$)
  ant#:=0
  iord#:=FALSE
  FOR i#:=1 TO LEN(s$) DO
    IF s$(i#:i#)=" " THEN
      iord#:=FALSE
    ELIF NOT iord# THEN
      iord#:=TRUE
      ant#:+1
    ENDIF
  ENDFOR i#
  RETURN ant#
ENDFUNC ordantal#

// Testprogram
DIM s$ OF 80
PRINT "Skriv en text på högst 80 tecken:"
INPUT AT 0,0,80: ":" s$
PRINT "Texten innehöll";ordantal$(s$);"ord."

// Övning 4-4
// Huvudprogram
INPUT "Ange ett heltal: ": dectal#
INPUT "Ange bas (2-36): ": bas#
PRINT "Talet blir: ";nybas$(dectal#,bas#)

FUNC nybas$(t#,b#) CLOSED
  DIM ny$ OF 31, siffror$ OF 36
  siffror$:="0123456789ABCDEFGHIJKLMNPOQRSTUVWXYZ"

  ny$:=""
  REPEAT
    p#:=t# MOD b#+1 // Adderar 1 pga att 1:a posit i siffra$ är 1.
    ny$:=siffror$(p#:)+ny$
    t#:=t# DIV b#
  UNTIL t#=0
  RETURN ny$
ENDFUNC nybas$

// Alternativ lösning ("traditionell"):
FUNC nybas2$(t#,b#) CLOSED
  DIM ny$ OF 31, siffra$ OF 1

  ny$:=""
  REPEAT
    rest#:=t# MOD b#
    t#:=t# DIV b#
    IF rest#<10 THEN
      siffra$:=CHR$(rest#+ORD("0"))
    ELSE
      siffra$:=CHR$(rest#-10+ORD("A"))
    ENDIF
    ny$:=siffra$+ny$
  UNTIL t#=0
  RETURN ny$
ENDFUNC nybas2$

```

```

// Övning 5-1
// Huvudprogram
PAGE
PRINT TAB(30);"SLUMPTAL PÅ FIL"
PRINT
INPUT "Ange hur många slumpstal du vill ha: ": antal#
INPUT "Ange minsta och största värde på slumpstalen: ": min#,max#
skriv_fil(antal#,min#,max#)
läs_fil

PROC skriv_fil(antal#,min#,max#) CLOSED
  OPEN FILE 1,"SLUMPTAL.DAT",WRITE
  FOR i#:=1 TO antal# DO
    tal#:=RND(min#,max#)
    WRITE FILE 1: tal#
  ENDFOR i#
  CLOSE FILE 1
ENDPROC skriv_fil

PROC läs_fil CLOSED
  sum:=0
  antal#:=0
  OPEN FILE 1,"SLUMPTAL.DAT",READ
  WHILE NOT EOF(1) DO
    READ FILE 1: tal#
    PRINT tal#;
    sum:=sum+tal#
    antal#:+1
    IF antal# MOD 10=0 THEN PRINT
  ENDWHILE
  CLOSE FILE 1
  PRINT
  PRINT "Summan =";sum
  PRINT "Antal slumpstal =";antal#
ENDPROC läs_fil

// Övning 5-4
DIM f1$ OF 15, f2$ OF 15, prograd$ OF 200

PRINT "RADNUMRERING AV COMALPROGRAM"
INPUT "Namnet på den programfil som ska radnumreras: ": f1$
REPEAT
  INPUT "Namnet på den nya programfilen: ": f2$
UNTIL f1$<>f2$
OPEN FILE 1,f1$,READ
OPEN FILE 2,f2$,WRITE
radnr#:=0
WHILE NOT EOF(1) DO
  INPUT FILE 1: prograd$
  radnr#:=radnr#+10
  PRINT FILE 2: USING "#### ": radnr#;
  PRINT FILE 2: prograd$
ENDWHILE
CLOSE
PRINT "Klart!"

```

```
// Övning 5-5a
DIM namn$ OF 10

OPEN FILE 1,"MAPROV.DAT",WRITE
WHILE NOT EOD DO
  READ namn$,bet1#,bet2#,bet3#,bet4#
  WRITE FILE 1: namn$,bet1#,bet2#,bet3#,bet4#
ENDWHILE
CLOSE

DATA "Allan",3,3,2,2,"Berta",4,5,4,4,"Dagmar",3,4,4,5,"Elsa",3,3,5,4
DATA "Erik",2,2,1,2,"Gunnar",4,3,2,1,"Harald",3,3,3,3,"Inga",2,1,1,2
DATA "Ivar",4,5,5,4,"Lena",4,4,5,5,"Mikael",2,2,2,3,"Nils",3,4,3,5
DATA "Nina",5,5,5,5,"Olof",3,3,3,3,"Peter",4,3,3,3,"Rune",3,3,4,4
DATA "Siv",3,5,5,4,"Stig",3,3,2,3,"Tina",2,2,3,3,"Ulf",1,2,1,1

// Övning 5-5b
DIM namn$ OF 10, bet#(4)

PAGE
PRINT "FEMMOR"
PRINT
REPEAT
  INPUT "Vilken skrivning vill du titta på (1-4): ": nr#
UNTIL nr#>=1 AND nr#<=4
OPEN FILE 1,"MAPROV.DAT",READ
PRINT
PRINT "Följande elever hade betyg 5 på skrivning";nr#,":";
WHILE NOT EOF(1) DO
  READ FILE 1: namn$
  FOR i#:=1 TO 4 DO
    READ FILE 1: bet#(i#)
  ENDFOR i#
  IF bet#(nr#)=5 THEN PRINT namn$;
ENDWHILE
CLOSE
```

```
// Övning A-1
PAGE
PRINT "SORTERINGSTEST"
PRINT
REPEAT
  INPUT "Ange antal heltal som ska sorteras: ": antal#
UNTIL antal#>2
DIM tal#(antal#)

// Generera och skriv ut osorterade slumpstal
PRINT "Osorterad lista:"
FOR i#:=1 TO antal# DO
  tal#(i#):=RND(0,2000)
  PRINT USING "#####": tal#(i#),
ENDFOR i#
PRINT
PRINT

TIMER 0 // Nollställ klockan

shellsort(antal#,tal#()) // Byts ut för andra sorteringsmetoder

tid:=TIMER
PRINT "Tidsåtgång: ";tid

// Skriv ut sorterad lista
PRINT "Sorterad lista:"
FOR i#:=1 TO antal# DO
  PRINT USING "#####": tal#(i#),
ENDFOR i#
```


NOTERINGAR

Sakregister

Ada 2 (ex)

Ada 2

Aktuell parameter 87

Alfanumeriska tangenter 6

Alt-tangent 6

AND 75

Aritmetiska operatorer 14

Aritmetiska uttryck 15

ASCII-kod 70, 121

AT 69

AUTO 40, 107

BackSpace 6

BASIC 1

Binära tal 105

Binärfiler 93

Binärsökning 102

Boolska konstanter 73

Boolska variabler 73

BYE 8, 107

C 2

Caps Lock 6

CAT 107

CHAIN 100

CHANGE 81

CLOSE 93

CLOSED 83, 87

CON 107

CON: 23

CREATE 95

Ctrl-tangent 6

CURSOR 22

Datatyper 30, 44

boolska 73

flyttal 30

heltal 30

numeriska 76

DEL 19, 107

Del-tangent 9

DELETE 24, 107

Delsträngar 51

DIM 33

DIR 24, 93, 107

Direktfiler 95

DISPLAY 81, 93, 107

DIV 76

EDIT 107

ELIF 71, 72

ENTER 81, 108

Enter-tangent 6

Esc-tangent 6

Euklides algoritm 90

EXIT 79

Exponentform 31

FALSE 73

Felhantering 103

Felmeddelande 10, 27

FILEFREE 91

Filhantering 93

Filnamn 91, 93, 95

Filtypsbeteckning 23, 93

Filtypstillägg 23, 91, 95

FIND 81

Flyttal 30, 76

Flyttalsområde 31

Flyttalsvariabler 30

Formell parameter 87

FREEFILE 91

Funktioner 69, 88

boolska 85

Funktionstangenter 19

GLOBAL 82, 84

Global variabel 87

Grafik 57, 82, 105

Grafikinstruktioner 57

Heltal 76

Heltalsvariabler 30

Hexadecimala tal 105

Home-tangenten 11

Hårddisk 7, 24

IBM-ASCII 83, 121

Identifierare 28

IF-satsen 71

IMPORT 87

IN 74

Indexerade variabler 82, 88

Infogningsläge 12

Inparameter 87

INPUT 36

INPUT AT 38

INPUT FILE 94, 95

Ins-tangent 9

INT 76

KEY\$ 70

Kommatecken 47

Kontrolltangent 6

LIST 18, 81, 93, 108

LOAD 25, 108

Logiska konstanter 73

Logiska operatorer 75

Lokal variabel 87

LOOP 79

Lämna COMAL 8

Mantissa 31

Markör 8

MERGE 81
MOD 76
Modula-2 2

NEW 17, 108
NOT 75
Numeriska variabler 33

Omdirigering 101
Omnumrera 22
OPEN FILE 93, 95
OR 75

PAGE 21
Parametrar 86
Pascal 2
PASS 108
Piltangenterna 9
Postlängd 95
PRINT 8
PRINT AT 21, 47, 69
PRINT FILE 94, 95
PRINT USING 45
Prioritet 15
Prioritetsregler 15
PRN: 23
Proceduranrop 86
Procedurer 86
Procedurhuvud 86
Procedurkropp 86
Procedurnamn 86
Program 17
Programnamn 23
Programsatser 17

Radnummer 17, 40
READ FILE 94, 95
Redigeringstangenter 9
REF 87
Referensanrop 87
Referensparameter 87
RENUM 22, 108
Reserverade ord 29
RETURN-satsen 88
Return-tangent 6
RUN 19, 109

SAVE 24, 93, 109
Sekventiella filer 93
SELECT 109
SELECT OUTPUT 23, 101
Semikolon 13, 47
Shellsortering 101
SIZE 109
Skift-PrtSc 23
Skifttangent 6
Skrivare 23
Skärmdump 23

Sortering 101
Starta COMAL 7
Starta grafik 57
Strängar 8
Strängdimensionering 33
Stränghantering 50
Strängvariabler 33
Sökning
 binär 102
 linjär 80, 98, 102

TAB 40, 48, 69
Tab-tangent 6
Tangentbordet 6
Textfiler 93
Tidmätning 101
Tilldelning 44
Tilldelningsoperatör 28, 44
Tilldelningssatsen 28, 44
TIMER 101
TRAP 103
TRUE 73
TYPE 93

Underprogram 86
UNIT 25, 109
Utparameter 87
Utvidgad ASCII 121

VAL 70
Variabel 27
Variabelnamn 28
Värdeanrop 88
Värdeparameter 87

WRITE FILE 94, 95

ZONE 47, 91

Överskrivningsläge 12