Fag forum – matematik nov. 1978 1.

Studielektor Børge R. Christensen, Danmark:

COMAL (Struktureret Basic)

Det begyndte i 1972, da vi fik en NOVA minidatamat her på den pædagogiske højskole i Tønder. Vi begyndte at programmere løs i Basic, på samme måde som man gjorde ved de fleste andre skoler, hvor man havde været så heldig at få en datamat. I begyndelsen gik det helt fint. Basic er let at lære, og både studenterne og jeg skrev en masse små programmer – mest af matematisk art – og de kørte fint. Men efterhånden blev programmerne større og fejlene hyppigere. Det skete ofte, at jeg måtte sidde temmelig længe for at finde fejlen i et program, der ikke virkede som det skulle. Det begyndte at irritere mig, at det ofte var vanskeligt at læse selv ret simple Basic-programmer.

Jeg fandt, at der er to hovedårsager til, at Basic-programmer er så vanskelige at læse og rette: De variables navne er for korte til at vise, hvad de står for, og de mange GOTO'er gør det vanskeligt og tidsrøvende at finde ud af, hvorledes de forskellige programafsnit hænger sammen og er opbygget. Lad os se på et meget simpelt eksempel:

```
0070  IF S<100 THEN 100
0080  PRINT «DE SKAL BETALE»;S;«KR.»
0090  GOTO 110
0100  PRINT «DE SKAL BETALE (INCL. GEBYR)»;S+15;«KR.»
0110  REM (GEBYR ELLER IKKE, DET VAR SAGEN)
```

Som det ses, er der to alternativer. Hvis S (salgsprisen) er mindre end 100, skal man betale et ekspeditionsgebyr på 15 kr. for at få ordren ekspederet. Hvis salgsprisen er 100 kr. eller mere, skal man ikke betale gebyr. Nu sker det ofte, at GOTO i linje 90 bliver glemt, og jo større programafsnittet mellem IF-sætningen og «brudpunktet» bliver, desto større er risikoen for, at det glemmes. Jeg synes også, at det er temmelig tåbeligt, at hvis det Boolske udtryk (åbne udsagn) i IF-sætningen er *sand* (fx. hvis S er mindre end 100), skal man *gå et andet sted hen* i programmet i stedet for at fortsætte med den eller de sætninger, der følger umiddelbart efter styresætningen. Det mest naturlige ville dog være, at man i *bekræftende* fald udførte det nærmeste job og først begynder at se sig om efter et *alternativ*, hvis udsagn er falsk. Og hvorfor skal salgsprisen være afgivet ved et S, eller hvis man er gavmild med fx. S1, og ikke med SPRIS, SALGSPR eller lignende, som ikke ville efterlade nogen tvivl om, hvad programmøren mener. I store programmer med mange variable kan man hurtigt tabe orienteringen med de alt for korte Basic-navne. Hvorfor kan programmet ovenfor ikke simpelthen skrives fx. sådan:

```
0070  IF SALGSPR>=100 THEN
0080    PRINT «DE SKAL BETALE»;SALGSPR;«KR.»
0090  ELSE
0100    PRINT «DE SKAL BETALE (INCL. GEBYR)»;SALGSPR+15;«KR.»
0110  ENDIF (GEBYR ELLER IKKE, DET ER SAGEN)
```

I en meget berømt bog, *The Elements of Programming Style* af Kerninghan og Plauger står der bl. a.: *«Skriv, hvad du mener simpelt og ligefremt»* og *«vælg variabelnavne, der ikke kan misforstås»*. Disse to simple og grundlæggende regler for programmering kan ikke bruges i Basic!

På den anden side er der elementer i Basic, som er særdeles anvendelige. Jeg mener, at *interaktiviteten* og den *dynamiske editor* er meget nyttige for anvendelsen til elementære programmer. Ligeledes er I/O sætningerne lette at bruge og ganske effektive. Og så *er* Basic let at lære. Jeg diskuterede sagen med flere kolleger ved datalogisk institut på Aarhus Universitet, og sammen med én af dem, Benedict Løfstedt, definerede jeg nogle udvidelser af Basic, så man kunne bruge det til at skrive programmer, der var lettere at læse og rette. Vi brugte de algoritmiske strukturer fra programmeringssproget Pascal, et Algol-lignende sprog, der er defineret af professor Niklaus Wirth fra ETH i Schweitz. Jeg ønskede at vort sprog skulle være en udvidelse of Basic af to grunde: De Basic-programmer, vi allerede havde, skulle stadig kunne køre under det nye system, og — som allerede nævnt — der er elementer i Basic, vi gerne ville beholde. Efter at udvidelserne var fastlagt, lykkedes det to af mine studenter, Knud Christensen og Per Christiansen, at ændre på koden for den Basic-fortolker, vi brugte — DGC Extended Basic — så de beskrevne udvidelser virkede. Vi kaldte vort nye sprog Comal (COMmon Algorithmic Language). Der er nogle, der mener, vi burde kalde det Struktureret Basic, men vi mener vore ændringer er så væsentlige, at sproget fortjener et nyt navn, der ikke er belastet med Basic-navnet.

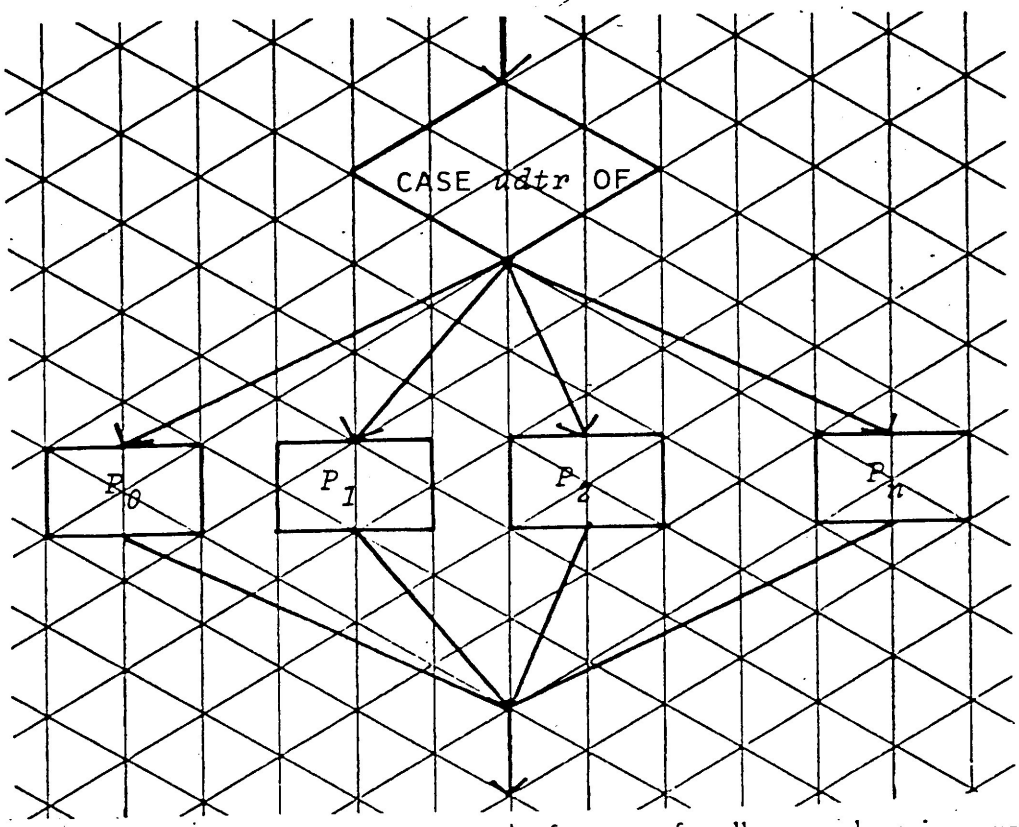
Jeg vil ikke trætte læseren med lange teoretiske udredninger om Niklaus Wirth og E. W. Dijkstras «structured programming» og «algorithmic structures». I stedet vil jeg gå lige til sagen og demonstrere sproget og principperne for vore udvidelser ved hjælp af et eksempel. Prøv at følge mig. Jeg refererer fra nu af til det programeksempel, der findes som bilag til artiklen. Programmet har naturligvis et *hoved* med nogle kommentarer om titel, forfatter, tidspunkt osv. (10—40). Derefter følger et *Hovedprogram* (60—330), der indledes med nogle erklæringer og et par tildelinger (70—80). I disse linjer kan man finde eksempler på vor udvidelse af Basic. I Comal kan man have *variabelnavne på op til 8 tegn*. Det første tegn skal være et bogstav, de følgende kan være bogstaver eller cifre. Som allerede nævnt er dette meget vigtigt, idet det er stærkt medvirkende til at gøre programmerne lettere at læse og forstå. Man kan også se, at tekstvariable markeres som i Basic ved, at der føjes et $-tegn efter navnet, som iøvrigt følger de samme regler for navne som tal-variable. I linje 70 erklæres også en *tekst-tabel* ELEVKART$, der kan indeholde op til 100 tekster med hver op til 30 tegn. Tekst-tabeller er helt uundværlige i et sprog, der bl. a. skal benyttes til elementære eksempler på informationssystemer. Bemærk også, at man i en LET-sætning kan have flere tildelinger (så mange, som linjens længde tillader), idet de enkelte tildelinger skal adskilles ved et semikolon. Iøvrigt består hovedprogrammet af en REPEAT.. UNTIL-løkke, fra hvis indre den egentlige styring foregår.

Jeg vil først forklare denne styrestruktur og så vende tilbage til REPEAT..UNTIL-løkker senere. INPUT-sætningen i linje 100 bevirker at den variable JOBKODE tildeles en talværdi af operatøren. Denne værdi overtages derpå af CASE-strukturen (110—310), der virker således: Fortolkeren undersøger de tilhørende WHEN-sætninger for at se efter, om den værdi JOBKODE har, findes efter et WHEN eller ej. Hvis operatøren har indtastet fx. 3, søger fortolkeren efter denne værdi og finder den efter WHEN i linje 190. *Nu udføres de linjer, der står mellem dette WHEN og det efterfølgende*, og fortolkeren går derpå videre med linjen, der følger umiddelbart efter ENDCASE. Hvis der indtastes en ulovlig kommando, fx. 8, udføres det afsnit, der er anført umiddelbart efter CASE-sætningen.

CASE-strukturen kan i almindelighed beskrives således:

```
CASE udtr OF

  [----------]
  [   P0     ]

WHEN list1

  [----------]
  [   P1     ]

WHEN list2

  [----------]
  [   P2     ]

  ...

WHEN listn

  [----------]
  [   Pn     ]

ENDCASE
```

og virker således:



Først bliver *udtr* (der betyder en konstant, en variabel eller et udtryk) evalueret, og fortolkeren begynder at undersøge værdierne på listerne efter de tilhørende WHEN-sætninger. Hvis værdien findes, bliver programafsnittet P_i mellem det aktuelle WHEN og det efterfølgende WHEN (eller ENDCASE) udført. Hvis det ikke bliver fundet, udføres programafsnittet P_0 lige efter CASE-sætningen. Når ét af de nævnte afsnit er udført, fortsætter fortolkeren med sætningen umiddelbart efter ENDCASE.

De enkelte lister efter WHEN-sætningerne kan indeholde lige så mange værdier, som linjens længde tillader (denne facilitet er ikke udnyttet i demonstrationsprogrammet). Listen kan også indeholde *udtryk* (såvel *aritmetiske* som *Boolske*) og hvis sådanne findes, udregnes de under søgningen. Dette giver nogle meget interessante muligheder. Prøv fx. at studere dette program nærmere:

```
0040  INPUT «ELEMENTET:», X
0050  CASE X OF
0060    PRINT «ELEMENTET ER IKKE I NOGEN AF MÆNGDERNE.»
0070  WHEN 2, 3, 5, 7, 11, 13, 17, 19
0080    PRINT «ELEMENTET ER I MÆNGDEN P.»
0090  WHEN 1, 4, 9, 16
0100    PRINT «ELEMENTET ER I MÆNGDEN K.»
0110  ENDCASE
```

Hvis det indtastede tal X har en værdi, der står i listen efter WHEN i linje 70, får man udskriften:

ELEMENTET ER I MÆNGDEN P.

Findes X derimod på listen efter WHEN i linje 90, får man udskriften:

ELEMENTET ER I MÆNGDEN K.

Findes værdien af X ikke på nogen af de nævnte lister, får man naturligvis udskriften:

ELEMENTET ER IKKE I NOGEN AF MÆNGDERNE

CASE-strukturen er også meget virkningsfuld, når man har med tekster at gøre. Prøv at betragte følgende:

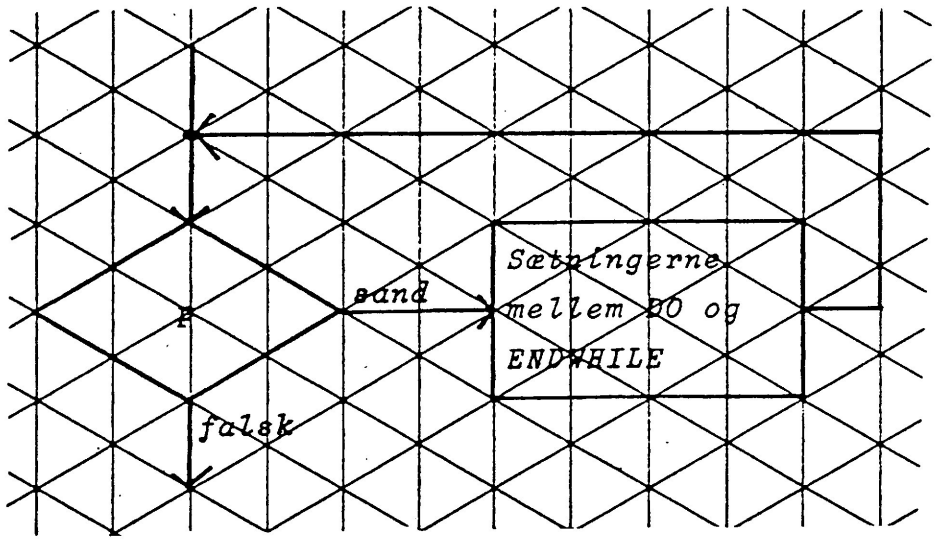
```
0040  INPUT «BOGSTAVET:», BOGST$
0050  CASE BOGST$ OF
0060    PRINT «DET ER EN KONSTANT.»
0070  WHEN «A», «E», «I», «O», «U», «Y», «Æ», «Ø», «Å»
0080    PRINT «BOGSTAVET ER EN VOKAL.»
```

CASE virker naturligvis også, selv om der kun er ét WHEN efter det.

Linjeindrykningerne, der skal fremhæve programmets struktur, udføres automatisk af fortolkeren, når programmet listes. Vi vil vende tilbage til dette senere. Man kan have lige så mange CASE-strukturer inden i hinanden, som man ønsker.

I hovedprogrammet har jeg også brugt EXEC (execute) sætningen. I linje 240 står det således: EXEC SLETNAVN. Dette er et *kald til et underprogram*, og vi kan lige så godt med det samme kaste et blik på det underprogram eller den *procedure*, der bliver kaldt. Man finder den i linje 630—750, og den begynder med sætningen PROC SLETNAVN og slutter med sætningen ENDPROC SLETNAVN. Procedurerne i Comal adskiller sig fra GOSUB i Basic ved, at der benyttes *navne* i stedet for linjenumre i kaldene og at procedurens begrænsninger tydeligt træder frem i Comal-listerne. Procedurer kan kalde nye procedurer i en dybde af indtil syv.

I linje 380—420 finder man endnu en Comal-struktur, nemlig WHILE..ENDWHILE-løkken. Den er meget simpel i sin virkning: Sålænge værdien af NAVN$ er forskellig fra «SLUT», gentages udførelsen af linje 390—410. Strukturen af WHILE..ENDWHILE er vist her:

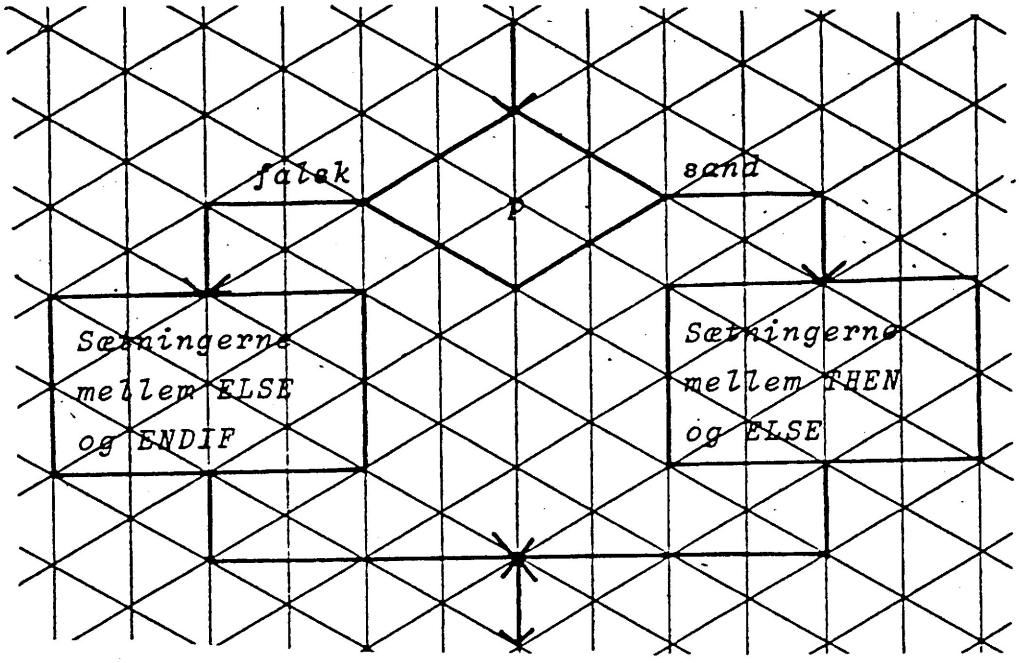


I diagrammet angiver *p* det Boolske udtryk, der styrer løkken.

IF..ENDIF-strukturen kan bedst demonstreres i PROC SLETNAVN (630—750). Lad os kaste et blik på linjerne 660, 720 og 740. Her finder vi nøgleordene IF, ELSE og ENDIF. Det hele styres af sætningen:

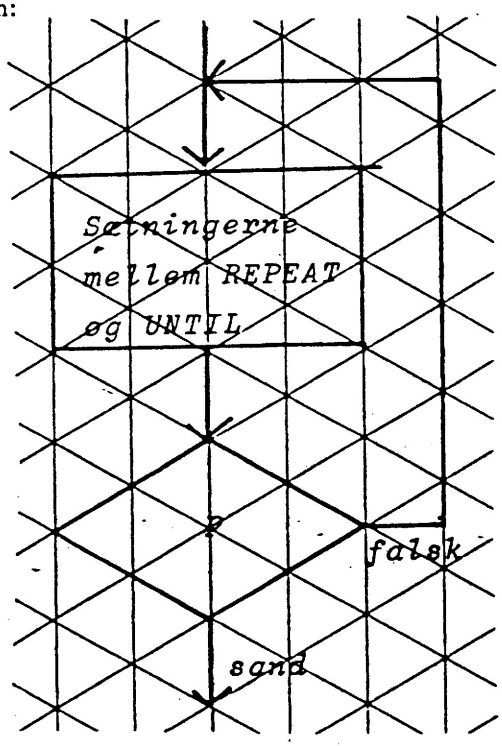
```
IF FUNDET THEN
```

i linje 660. FUNDET er en variabel, der bruges som et Boolsk udtryk (altså faktisk en *Boolsk variabel*). Den bliver fortolket som *sand*, hvis den har en værdi, der er *forskellig fra nul*, og som *falsk*, hvis den har *værdien nul*. Dette er i grunden ikke en Comal-facilitet, men den fandtes allerede i den Basic-version, vi byggede på. Hvis FUNDET fortolkes som *sand*, udføres linje 670—710, og hvis den fortolkes som *falsk*, udføres linje 730. Strukturen af IF..ELSE..ENDIF vises her:



Udtrykket *p* kan naturligvis være et hvilket som helst Boolsk udtryk (indbefattet Boolske konstanter og variable).

Indtil nu har jeg ikke forklaret noget om REPEAT..UNTIL. Lad os se på PROC SPORHUND (900—960). I linje 920 og linje 950 kan man finde REPEAT- og UNTIL-sætningerne. Strukturen er den mest selvindlysende, vi har indført, men her er ikke desto mindre dens diagram:



Lad os se litt nærmere på det indre af REPEAT..UNTIL-løkken. Vi finder dels en tæller NR, der bruges til at udpege navnene i kartoteket ét for ét. I linje 940 står der:

```
FUNDET=(NAVN$=ELEVKART$(NR))
```

Lad os læse den fra højre mod venstre. ELEVKART$(NR) er det navn, der har nummer NR i kartoteket. Dette navn sammenlignes med det indtastede navn, der står i NAVN$. Hvis de to navne er ens, får FUNDET tildelt værdien 1 (altså *sand*), og ellers får den tildelt værdien 0 (*falsk*). Sammenligningen mellem det indtastede navn og navnene i kartoteket fortsætter

```
UNTIL FUNDET OR NR=MAXNR
```

altså indtil navnet er fundet, eller vi har bladet hele kartoteket igennem. I alle tilfælde vil FUNDET altså have værdien *sand*, hvis navnet er fundet, og værdien *falsk*, hvis det ikke er tilfældet. REPEAT..UNTIL har vist sig at være den mest populære struktur hos fx. børnene i folkeskolens ældste klasser.

IF..ELSE..ENDIF, WHILE..ENDWHILE og REPEAT..UNDTIL kan hver for sig og uafhængig af hinanden indlejres i en dybde af syv. Hvis man altså bruger dem sammen med gode, gamle FOR..NEXT, kan man indlejre styrestrukturer i hinanden til en dybde af otte-og-tyve. Forfatteren mindes ikke at have set dette i praksis.

Ethvert program, der er skrevet i DGC Ex-

tended Basic kan udføres af vor Comal fortolker. Det betyder, at vort bibliotek og de programmer, vi får udefra i Basic, kan bruges med meget små eller slet ingen ændringer. Hvis det er nødvendigt at ændre dem, skyldes det ikke Comal, men Basic. Som bekendt er der efterhånden store forskelle på de forskellige Basic-versioner.

Dette, at Comal-fortolkeren selv rykker linjer, der styres fra andre linjer, ind, når der skrives en programliste, har vist sig at være vigtigere, end vi forudså, da vi indførte det. Det virker som en slags «global fejlmelding». Lad os forestille os, at vi har glemt at lukke en IF-forgrening med ENDIF. Man kan da straks af programlisten se, at der er noget galt, eftersom sætningerne ikke «slutter op» mod linjenumrene ved afslutningen af listen. Desuden — og det er meget vigtigt — s[illegible]es de studerende at blive mere bevidste om programstrukturen, når de anvender Comal.

Det har også glædet os, at vore strukturer har vist sig meget nyttige ved skrivning af programmer til datamat-støttet undervisning (CAI). Det tænkte vi ikke så meget på, da vi definerede Comal, men det har vist sig at være tilfældet. Især bruges CASE-strukturen meget af vore kolleger, der arbejder med CAI.

Vi har nu brugt Comal i ca. tre år, og sproget er næsten enerådende på seminarier og gymnasier her i Danmark. Jeg undrer mig ofte over, at der stadig er lærere, som er tilfredse med Basic. Sproget var måske godt nok i 1967, men det er mere end 10 år siden! Og prøv at se på udviklingen siden da. Minidatamaten var lige netop opfundet i 1967. Ydermere er det ikke nogen overvældende stor sag at skrive en Comal-fortolker. Vi behøvede kun at føje ca. 12 % til Basic-fortolkeren for at få Comal til at køre. En meget fin Comal-version er netop blevet færdig til mikrodatamaten 8080, og vi arbejder selv med at skrive en til Z80 mikrodatamaten. Denne sidste bliver iøvrigt en meget stærk version med parametre i forbindelse med procedurerne og lokale variable.

Jeg vil opfordre mine norske kolleger til ikke uden videre at godtage det programmel, datafirmaene vil sælge. Hvorfor skal jeres studenter nøjes med programmel, baseret på principper fra 1957 (Fortran), når skolerne køber udstyr fra 1978? Fra Dijkstra, Wirth, Hoare og jeres egne berømte dataloger fra Norsk Regnecentral ved vi i dag meget mere om, hvordan gode programmeringssprog skal være opbygget, end man gjorde i 1957 og — delvis — i 1967. Denne viden bør vi naturligvis bruge, og ikke lade os afspise med andenrangs programmel fra amerikanske colleges. Stil jeres krav *nu*, inden i får købt for mange systemer til skolerne!

Bilag:

```
*LIST
00010 REM ** PROGRAM: NUSSESKOLENS ELEVKARTOTEK **
00020 REM ** SKREVET FOR 'MATEMATIKK FAGFORUM' **
00030 REM ** AF BØRGE R. CHRISTENSEN **
00040 REM ** DENNE VERSION ER AF 7. AUGUST 1978 **
00050 REM //---------------//
00060 REM ** HOVEDPROGRAM **
00070 DIM ELEVKART$(100,30),NAVN$(30),TEGN$(1)
00080 LET NR=0; MAXNR=0
00090 REPEAT ** KARTOTEKET ER I BRUG **
00100   INPUT "1=IND/2=UD/3=SØG/4=SLET/5=SORT/6=STOP: ",JOBKODE
00110   CASE JOBKODE OF
00120     PRINT "SAADAN NOGET LAVER JEG IKKE!"
00130   WHEN 1
00140     REM ** INDTASTNING AF NAVNE **
00150     EXEC NAVNEIND
00160   WHEN 2
00170     REM ** UDSKRIFT AF KARTOTEKET **
00180     EXEC OVERSIGT
00190   WHEN 3
00200     REM ** SØGNING AF NAVN **
```

```
0210      EXEC FINDNAVN
0220    WHEN 4
0230      REM ** SLET ET NAVN **
0240      EXEC SLETNAVN
0250    WHEN 5
0260      REM ** ALFABETISK SORTERING AF NAVNENE **
0270      EXEC ALFASORT
0280    WHEN 6
0290      REM ** NU PIBER ALLE FLØJTER: FYRAFTEN **
0300      STOP
0310    ENDCASE
0320  UNTIL 0
0330  END ** SLUT HOVEDPROGRAM **
0340  REM //---------------//
0350  PROC NAVNEIND
0360    LET NR=MAXNR
0370    INPUT "ELEVENS NAVN (EFTERNAVN, FORNAVN(E)): ",NAVN$
0380    WHILE NAVN$<>"SLUT" DO
0390      LET NR=NR+1
0400      LET ELEVKART$(NR)=NAVN$
0410      INPUT "> ",NAVN$
0420    ENDWHILE ** IKKE FLERE NAVNE **
0430    LET MAXNR=NR
0440  ENDPROC NAVNEIND
0450  REM //---------------//
0460  PROC OVERSIGT
0470    REM ** UDSKRIFT AF KARTOTEKET **
0480    FOR NR=1 TO MAXNR
0490      PRINT ELEVKART$(NR)
0500    NEXT NR
0510  ENDPROC OVERSIGT
0520  REM //---------------//
0530  PROC FINDNAVN
0540    INPUT "HVILKET NAVN SØGES: ",NAVN$
0550    EXEC SPORHUND
0560    IF FUNDET THEN
0570      PRINT "ELEVEN STAAR SOM NUMMER";NR;"I KARTOTEKET."
0580    ELSE
0590      PRINT "VI HAR IKKE NOGEN ELEV AF DET NAVN."
0600    ENDIF ** AT VAERE ELLER IKKE VAERE **
0610  ENDPROC FINDNAVN
0620  REM //---------------//
0630  PROC SLETNAVN
0640    INPUT "HVILKET NAVN SKAL SLETTES: ",NAVN$
0650    EXEC SPORHUND
0660    IF FUNDET THEN
0670      LET MAXNR=MAXNR-1
0680      FOR J=NR TO MAXNR
0690        LET ELEVKART$(J)=ELEVKART$(J+1)
0700      NEXT J
0710      REM ** NAVNET ER SLETTET **
0720    ELSE ** NAVNET VAR DER IKKE **
```

```
0730     PRINT "VI HAR IKKE NOGEN ELEV AF DET NAVN."
0740   ENDIF ** SLETTET ELLER IKKE FUNDET **
0750 ENDPROC SLETNAVN
0760 REM //---------------//
0770 PROC ALFASORT
0780   REM ** BOBLESORTERING **
0790   FOR I=1 TO MAXNR-1
0800     FOR J=I+1 TO MAXNR
0810       REM ** HVIS DET I'TE NAVN KOMMER EFTER DET J'TE **
0820       IF ELEVKART$(I)>ELEVKART$(J) THEN
0830         REM ** BYT NAVNENE OM **
0840         LET NAVN$=ELEVKART$(I); ELEVKART$(I)=ELEVKART$(J); ELEVKART$(J)=NAVN$
0850       ENDIF ** OMBYTNING SLUT **
0860     NEXT J
0870   NEXT I
0880 ENDPROC ALFASORT
0890 REM //---------------//
0900 PROC SPORHUND
0910   LET NR=0
0920   REPEAT ** SØG EFTER NAVNET **
0930     LET NR=NR+1
0940     LET FUNDET=(NAVN$=ELEVKART$(NR))
0950   UNTIL FUNDET OR NR=MAXNR
0960 ENDPROC SPORHUND
0970 REM //---------------//
0980 REM ** SLUT NUSSESKOLENS ELEVKARTOTEK **
```

Harald Opheim:

Bedre programmeringsvaner!

Vær systematisk.

Profesjonelle EDB-folk legger nå større vekt på systematikk og dokumentasjon av programsystemer enn før. I ei tid hvor maskinkostnadene går ned mens lønningene stiger, er det viktig å lage programmene slik at arbeidet med å rette feil blir så lite som mulig og at en mulig programutvidelse kan foretas uten drastiske ombygginger. I løpet av de siste åra er det utviklet en rekke metoder som skal forenkle sjølve programmeringen, men ingen av dem kan sies å være Metoden med stor M. I en rapport heter det lakonisk: «Det eneste sikre synes å være at metodeanvendelse gir resultater, å være systematisk er bedre enn å være usystematisk.» (Hygen 1978.)

Læring om detaljer.

Metodelæring synes i liten grad å ha slått igjennom i EDB-undervisningen i den videregående skole. Læreren tvinges til å fortelle om hvordan de enkelte programsetningene skal se ut, og får liten anledning til å gi tips om gode prinsipper.

Med det timetallet som er til disposisjon, er det sjølsagt ikke mulig å lære elevene alt om EDB (les programmering). Enkelte gode vaner bør imidlertid innarbeides. Hvis det forsømmes, taler mye for at elevene tar grunnleggende uvaner med seg videre, og det er synd for de som kommer til å drive programutvikling seinere i livet. Danske erfaringer er at «eleverne slavisk arbeider efter