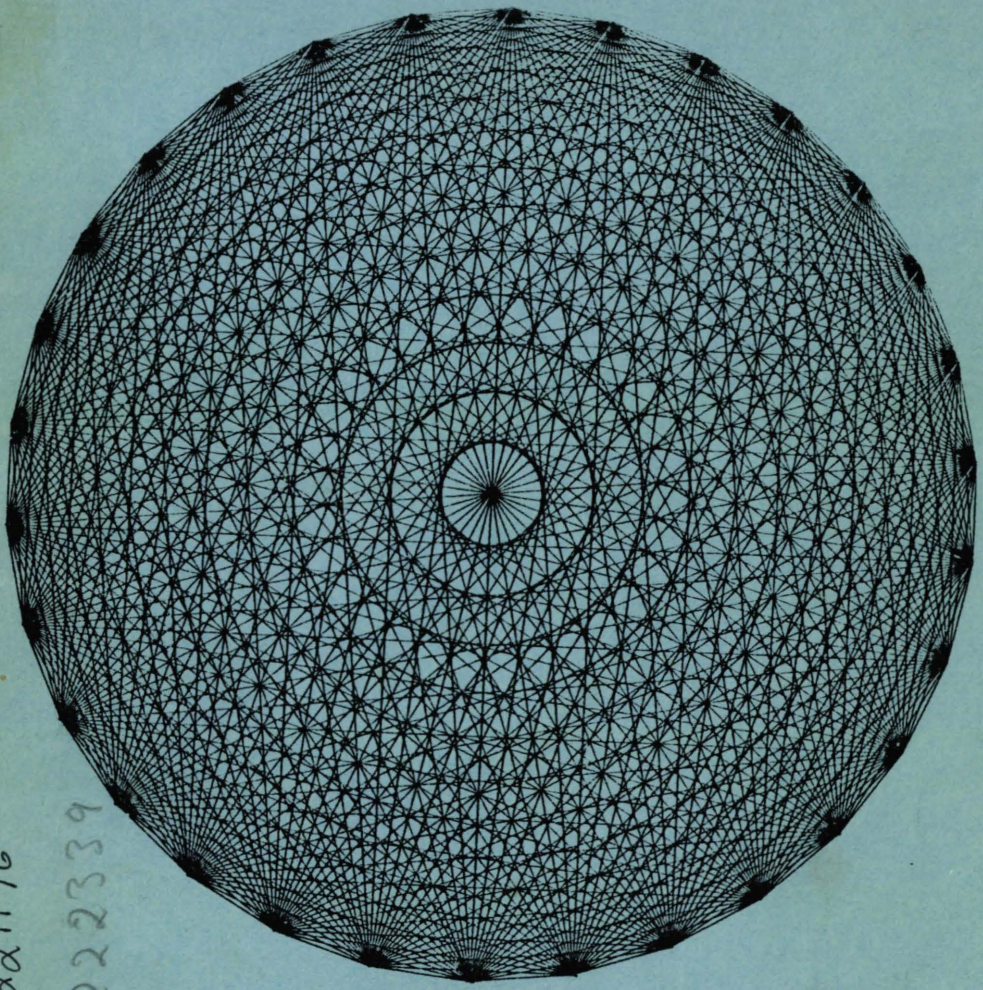
DaTS

Datalære ved de tekniske skoler



juni. 1983 nr. 2

Redaktion:

Ole Karmark
(ansvarshavende)

Helge Jensen

Kaj Thrysøe

Ekspedition:

DaTS
Århus Tekniske Skole
Halmstadgade 6
8200 Århus N

Programmet til udtegning af forside og programmer listet i bladet tilsendes interesserede, mod fremsendelse af formatteret diskette til DaTS.

Indhold

Artikler

Anmeldelser

Redaktionelt forord

Hermed foreligger det andet nummer af datalærebladet for de tekniske skoler. Den opmærksomme læser vil bemærke, at bladet har ændret navn fra DATA+ALGORITMER til DaTS, kort og godt. Navneskiftet ændrer ikke noget i bladets målsætninger, nemlig at virke som forum for udveksling af erfaringer og opdatering af viden om EDB hos lærerne på de tekniske skoler. Navneskiftet skal primært ses som en mere præcis overskrift for bladets fagområde: datalære ved de tekniske skoler. Samtidig håber vi også hermed at brede perspektivet ud for bladets faglige politik. Det bør ikke blot dreje sig om programmel og maskinel, selv om det naturligvis er fagets grundlag. -Vi bør også interessere os for de menneskelige konsekvenser af EDB-teknologien, og her tænker vi på EDB-baserede systemer mere generelt. Dette er iøvrigt også i tråd med formålsbeskrivelsen for faget datalære på de tekniske skoler. Dette nummer følger to tidligere artikler op fra første nummer. Den ene omhandler porte i mikrodatamater, og den anden drejer sig om algoritmebeskrivelse. Desuden er der en artikel om assemblerprogrammering, en artikel om billedbehandling og en mere teoretisk præget artikel om begrebsbrug. Endvidere har vi udbygget nummeret med en anmeldersektion.

Det er vort håb, at vi på længere sigt kan udarbejde egentlige temanumre og udvide bladet med en debatsektion. Imidlertid, skal vore mål nås kræver det en aktiv medvirken hos vore kolleger rundt om på de tekniske skoler. - Dette er hermed en opfordring til dem, der ligger inde med "noget", enten i skrivebordsskuffen eller i baghovedet. Skriv det ned og send det til redaktionen, det kunne have større interesse end man umiddelbart troede.

Red.

Mennesker og EDB:

Ved Kaj Thrysøe

Begrebsdannelse og begrebsbrug.

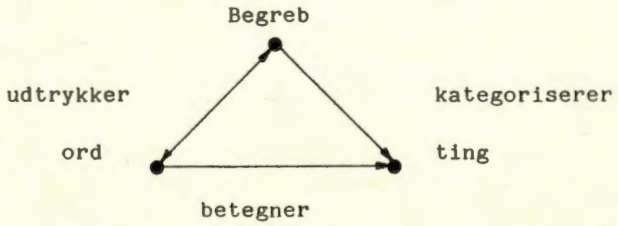
Artiklen vil beskrive begrebsudviklingsprocessen på forskellige planer og redegøre for forskellen mellem den begrebsdannelse og -brug, der foregår indenfor faget Datalogi, og den begrebsdannelse og -brug, der foregår i andre sociale sammenhænge.

Hensigten er at afdække forholdet mellem menneske og maskine, og i dette samspil har sproget en meget stor betydning. Artiklen hævder endvidere, at denne problemstilling bør være central i patalæreundervisningen på de tekniske skoler.

Begrebets struktur.

Et begreb indeholder dels sin egen betegnelse, og dels kategoriserer begrebet de størrelser, der falder ind under begrebet. (1)

Vi kan illustrere begrebsstrukturen grafisk:

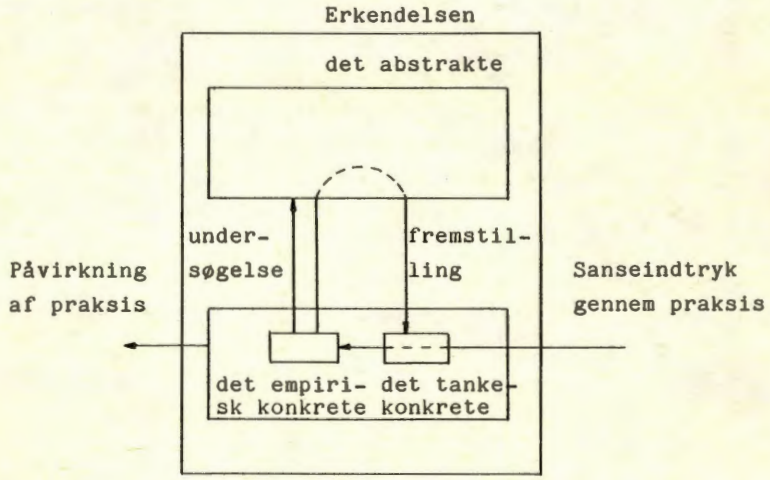


Strukturen angiver tre relationer med følgende indhold:

1. Ordet betegner tingen.
2. Ordet udtrykker begrebet, som igen giver ordet mening.
3. Begrebet kategoriserer tingen og bestemmer dermed det, der falder ind under begrebet.

Menneskelig erkendelse og begrebsdannelse.

Mennesket danner sine begreber i en stadig erkendelsesproces.(2) Udgangspunktet for enhver begrebsdannelse er den konkrete virkelighed, der -for mennesket- umiddelbart fremstår som en kaotisk forestilling om denne virkelighed. (Man kender sikkert fornemmelsen fra sit første møde med en datamaskine ?)
Denne umiddelbare forestilling gøres middelbar ved at indsnævre virkelighedsfeltet og tage enkelte dele op til nærmere undersøgelse. Tankeprocessen -eller erkendelsesobjektet er nu kommet på begreb og benævnes derfor det tankekonkrete.
Vi kan illustrere processens principielle forløb på følgende måde(3):



Denne tankeproces vil i princippet fortsætte med de enkelte genstande som objekt, indtil der etableres et samlet begreb eller begrebssystem om den virkelighed, der oprindelig fremstod som en kaotisk forestilling. Det usystematiske er blevet systematiseret ved hjælp af begrebsdannelserne, og denne proces har frembragt større viden og erkendelse om virkeligheden.

Begrebsdannelse.
Den klassiske logik.

Lars Mathiassen's teori om den generelle erkendelsesudvikling og begrebsdannelse er netop generel og derfor utilstrækkelig til at forstå specifikke forskelle i begrebsudvikling og -især- begrebsbrug. Lad os se lidt på forholdet mellem den klassiske logik's begrebsdannelse og den mere "naturlige" begrebsdannelse.(4)
Den klassiske, aristoteliske logik opererer med klart definerede begreber, der populært sagt er lukket i "begge ender".
Et begrebs indhold er den samling egenskaber eller træk, som begrebets eksemplarer har tilfælles. Begrebsindholdet kaldes også begrebets intension.
Et begrebs omfang omfatter den samling af genstande eller fænomener, der falder ind under begrebets indhold. Begrebets omfang kaldes også begrebets ekstension.
Et begrebs indhold fastsætter nogle definerende egenskaber, og er de ikke tilstede, falder genstanden udenfor det aktuelle begreb. Begrebets definerede egenskaber fastlægger således en skarp grænse mellem, hvad der falder "indenfor" h.h.v. "udenfor" begrebet; der er klart tale om et enten - eller.
Begreberne antages endvidere at være ordnet hierarkisk, ved opdeling og underopdeling af de mest brede og almene begreber. Således har et begreb højere oppe i hierarkiet altid en mere omfattende ekstension end dets underordnede begreber, men samtidig indeholder det overordnede begreb færre definerende egenskaber end hvert af de underordnede.
Denne opfattelse af begreber er meget udbredt indenfor naturvidenskaberne. Også indenfor Datalogien finder den klassiske begrebsmodel stor anvendelse, her tænkes især på mængdelæreren.

Det er almindeligt - indenfor struktureret programdesign- at opfatte programmets omfang som en mængde og programmets underprogrammer (procedurer) som delmængder heraf. Denne begrebsmetodik gør programmet mere overskueligt for systemdesigneren, og det har sine klare fordele - set fra konstruktørens synsvinkel.
Imidlertid medfører denne strukturering af virkeligheden i mængder og delmængder, at brugeren af systemet skal tænke i mængder - og det vel at mærke indenfor det univers systemdesigneren har defineret. Dette forhold vil være relativt uproblematisk blandt "fagfolk". -Men der bliver problemet for den bruger, der dels ikke ved, hvordan systemet strukturerer sine data og dels ikke er trænet i at tænke systematisk indenfor den aristoteliske tradition.
Problemet bliver så, at systemdesigneren og brugeren tænker ud fra forskellige præmisser, fordi deres begrebssystemer har forskellig karakter. Her ligger kimen til en konflikt mellem maskine og bruger.

Associativ begrebsdannelse.

Den klassiske logik's begrebsmodel er blevet draget i tvivl, fordi begreberne naturligt ikke er skarpe. I stedet for ægte mængder, klasser og kategorier er det måske rettere at karakterisere begreberne efter objekternes typiskhed, og benævne begreberne som prototyper.
Indenfor prototypebegrebet kan genstandene så underordnes begrebet i større eller mindre omfang, og det medfører at grænserne mellem begreberne bliver mere åbne "i enderne" eller flossede i kanterne"; der kan nu mellem begreberne blive tale om et både - og.
Den prototypiske begrebsdannelse er rimeligvis den fremgangsmåde de fleste mennesker anvender. Tesen er testet i en undersøgelse af børns sprogindlæring, og resultatet tyder på, at begrebsudviklingen faktisk foregår i relation til allerede kendte ting. Begreberne dannes ikke i et tomrum; men i relation til størrelser, der er kendte i forvejen.

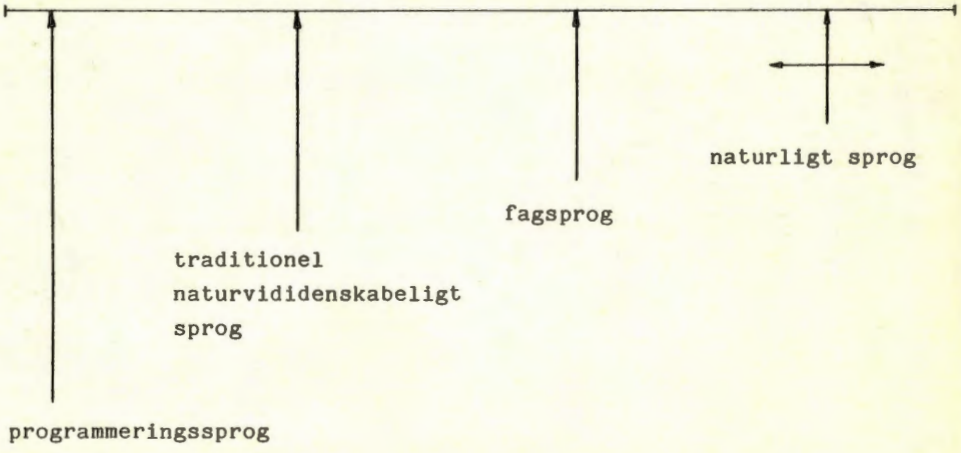
Begrebssystemerne kan endvidere være mere eller mindre veludviklede afhængigt af daglige sociale relationer og erfaringer. Denne tænkning kaldes associativ, fordi det nye kun kan forstås, hvis det kan associeres med noget gammel kendt.

Begrebernes betydning for kommunikation.

Den aristoteliske- og den associative begrebsdannelse kan siges at udgøre yderpunkterne på en skala over begrebernes forskellighed. Skala'en og den socialt betingede begrebsbrug kan illustreres med følgende tegning:

Aristoteliske begreber: (lukket begreb: fænomenernes medlemsskab afgørligt med ja - nej)

Prototype begreber: (åbent begreb: fænomenernes medlemsskab ikke altid afgørligt med ja - nej)

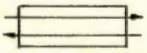


Begrebernes socialt bestemte forskellighed får konsekvenser for en given kommunikation. Hvis kommunikationen skal kunne foregå meningsfyldt, er det afgørende at afsender (taler) og modtager (lytter) kan forstå hinanden, d.v.s. have kendskab til hinandens erfaringsunivers. Lad os tydeliggøre problemet med tre konkrete eksempler:

1. Fri kommunikation uden begrænsende kanaler.

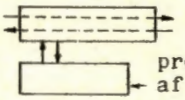
I denne situation vil både afsender og modtager kunne kommunikere verbalt og non-verbalt. Kan modtageren ikke forstå afsenderens budskab, vil dette kunne signalleres sprogligt (ved kommentarer) eller ikke-sprogligt (gaben, kigge den anden vej etc.)
Kommunikationen er ikke begrænset af den nationalsproglige syntaks, og derfor er sprogets formaliseringsgrad lav.

2. Kommunikationen er begrænset af en kanal -f.eks. en telefon eller en terminal.



I denne situation kan kommunikationen kun foregå sprogligt. Kommunikationen er begrænset af den nationalsproglige syntaks (og semantik). Kommunikationens formaliseringsgrad er steget i og med, at det ikke-sproglige element er fjernet. De kommunikationsmæssige problemer denne formalisering medfører kan let iagttages hos børn, der skal "lære" at tale i telefon. (Tænk også på EFG-eleverne, der første gang skal ringe ud til en virksomhed og indhente oplysninger om et eller andet emne?)

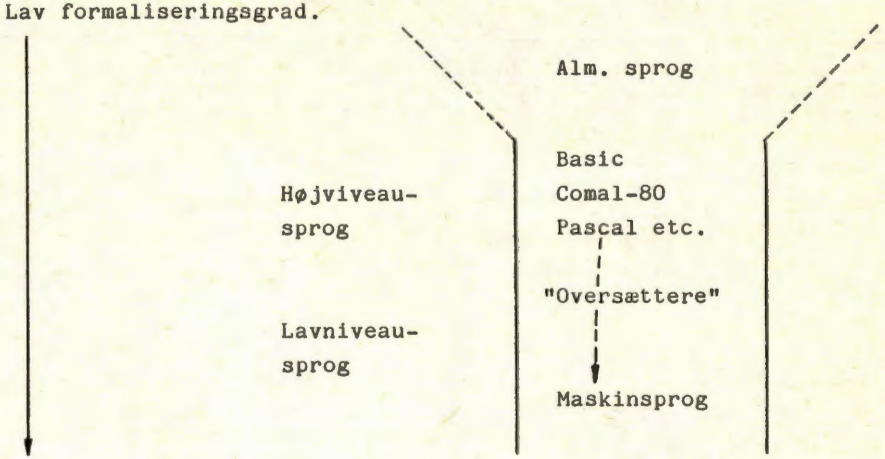
3. Kommunikation bearbejdet af en processor.



I denne situation kan kommunikationen ligeledes kun foregå sprogligt. -Men her er kommunikationen begrænset af datasproget's syntaks og semantik. Kommunikationens formaliseringsgrad er steget til det (næsten) absolutte, og vi befinder os nu indenfor den aristoteliske begrebsverden.

For systemdesigneren bliver denne indsigt om begrebernes forskellighed og sprogets formaliseringsgrad vigtig.
Systemdesigneren må planlægge systemet ud fra brugerens forudsætninger, d.v.s. brugerens erfaringsverden eller begrebsverden. Kan systemdesigneren ikke sætte sig ind i brugerens tankeverden, vil brugeren formentlig blive fremmedgjort overfor systemet. (tænk f.eks. på Postterminalen i Kbhn., og DR's berømte DORA, som medarbejderne har døbt om til MANUELLA)
Det betyder endvidere at lærerne i undervisningen skal inddrage denne problemstilling om forholdet mellem menneske og maskine. Derved vil meget mystik omkring EDB-maskiner og programmeringssprog forsvinde.
Afslutningsvis vil vi illustrere forholdet mellem menneske og maskine -sprogligt set- med følgende tegning:

Lav formaliseringsgrad.



Høj formaliseringsgrad.

Det begrebsmæssige og sproglige perspektiv er bestemt af Maskinsproget's form (kombinationer af 0 eller 1).

Litteraturhenvisninger.

1. Joachim Israel: Sociologisk Grundbog. Kbhn. 1978 s. 379
2. Lars Mathiassen: System og systemudviklingsmetode. Århus 1982
3. Figuren er gengivet efter Lars Mathiassen op. cit. s. 28
4. Resten af artiklen bygger især på Steen Folke Larsen: Egocentrisk tale, Begrebsstruktur og Semantisk udvikling.
   Nordisk Psykologi, 1980. s. 55-73.

Algoritmer til forstørrelse af billeder

Ved Lone Verner Nielsen

Dette indlæg er skrevet for at vise et eksempel på, hvordan edb-anvendelse fra arbejdspladserne kan anskueliggøres på algoritmeniveau, og hvordan man kan anvende Cometen til at simulere den pågældende edb-anvendelse. Det valgte eksempel er forstørrelse af digitalt lagrede billeder. Forhåbentlig opnås en bedre forståelse for - i dette tilfælde - scannerens virkemåde.

Eksemplet er hentet fra Grafo-projektets undervisningsmateriale: Grafisk Datalære (januar 1983), som jeg har været med til at udarbejde.

Forstørrelse i virkeligheden

På mange grafiske arbejdspladser anvendes en scanner ved repoduktion af billeder. Originalen placeres på en analysetromle, hvor sætningen registreres for hvert billedelement, og omsættes til en talværdi. De registrerede værdier gemmes i en tabel - et array- og der kan udføres forskellige manipulationer på de lagrede talværdier, afhængig af hvilke indstillinger operatøren har foretaget på betjeningspanelet, f. eks. størrelsesændring. Senere kan billeder "udskrives" f.eks. på en film.

Algoritmer til forstørrelse

Antag at vores lagrede billede ser således ud:

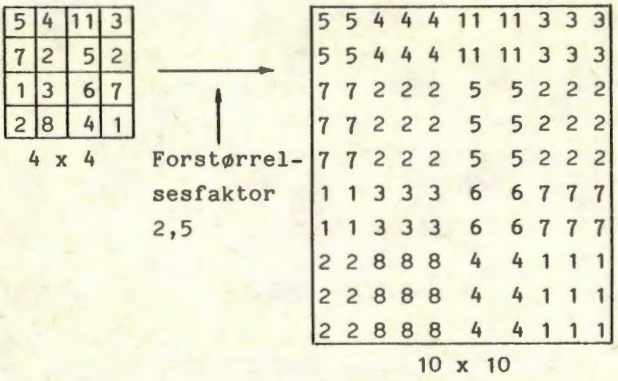
5	4	11	3
7	2	5	2
1	3	6	7
2	8	4	1

hvor tallene er et mål for den sværtning, der måltes i de enkelte billedelementer (det mørkeste område var omkring 3. billedelement i 1. række).

Den måde, hvorpå en scanner kan forstørre et digitaliseret billede, er ved at gentage de enkelte billedelementer i originalen i overensstemmelse med den ønskede forstørrelsesfaktor (en metode som dog ikke er god nok, hvis der er tale om store forstørrelser).

Vi skal derfor først have indlæst forstørrelsesfaktoren. Dernæst skal vi oprette et nyt, større array til opbevaring af sværtningsværdierne for det forstørrede billede.

Forstørrer vi ovenstående billede med en faktor 2,5 (svarende til 250%), skal hvert enkelt billedelement i originalen gentages 2 gange, og desuden skal hvert andet (10/5 = 2) billedelement gentages endnu en gang. Dvs:



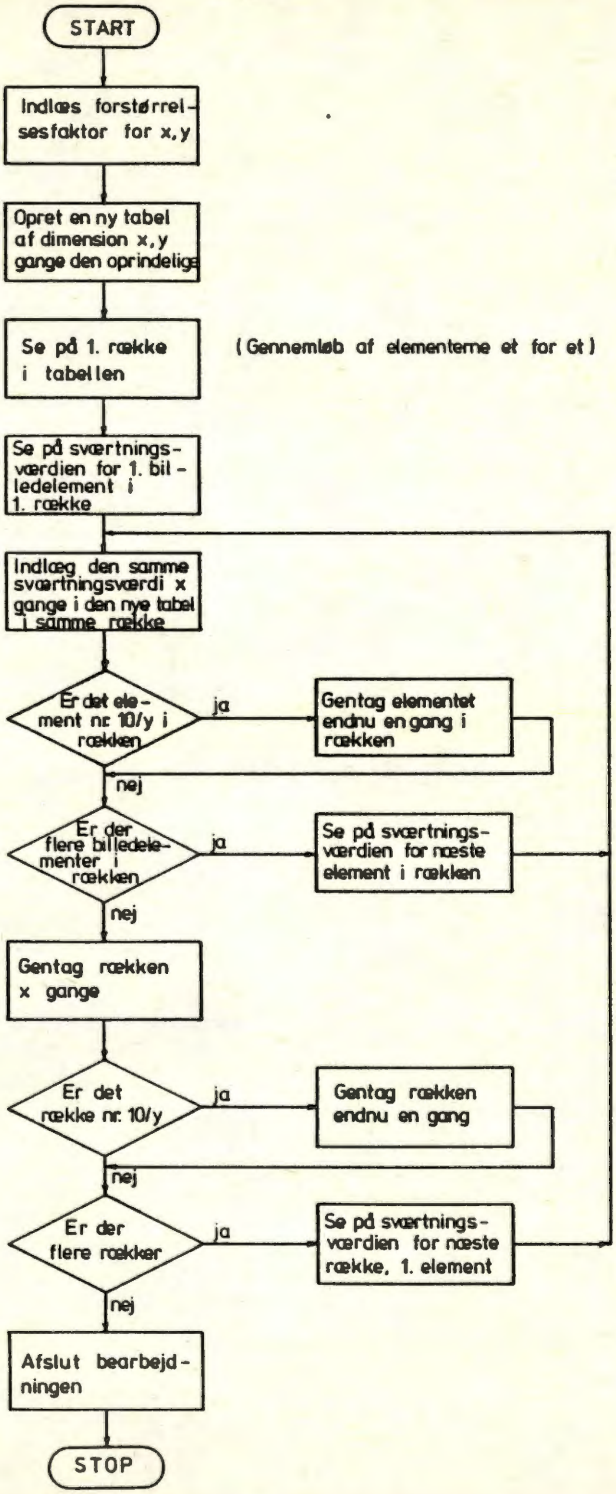
I nedenstående algoritme har vi begrænset os til forstørrelsesfaktorer x,y hvor

x angiver, hvormange gange det enkelte billedelement skal forekomme i det nye billede (det nye array)

y angiver, at y/10 af elementerne yderligere skal gentages en gang. Dvs. at hvert 10/y element skal gentages endnu en gang.

Selve forstørrelsen udføres ved at gennemløbe samtlige billedelementer systematisk (række for række).

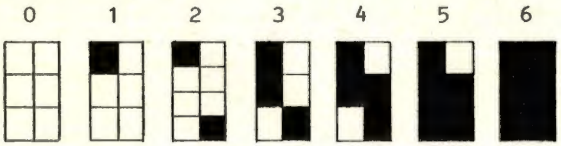
Forstørrelsesalgoritmen kan i sin helhed se således ud:


START
Indlæs forstørrelsesfaktor for x,y
Opret en ny tabel af dimension x,y gange den oprindelige
Se på 1. række i tabellen
(Gennemløb af elementerne et for et)
Se på sværtningsværdien for 1. billedelement i 1. række
Indlæg den samme sværtningsværdi x gange i den nye tabel i samme række
Er det element nr. 10/y i rækken
ja
Gentag elementet endnu en gang i rækken
nej
Er der flere billedelementer i rækken
ja
Se på sværtningsværdien for næste element i rækken
nej
Gentag rækken x gange
Er det række nr. 10/y
ja
Gentag rækken endnu en gang
nej
Er der flere rækker
ja
Se på sværtningsværdien for næste række, 1. element
nej
Afslut bearbejdningen
STOP

Forstørrelse på Cometen, algoritme

På Cometen kan vi på en simpel måde illustrere billedbehandlingen, ved at anvende de semigrafiske tegn og passende programmer.

Vi har ingen scanner til rådighed, så vi må for hvert billede indtaste dets størrelse samt sværtningsværdier for hvert enkelt billedelement. Vi har valgt sværtningsværdierne 0 - 6, og vi har valgt, at de skal udskrives som flg. semigrafiske tegn:



Et specielt udskrivningsprogram udskriver "sværtningsværdierne" i de enkelte billedelementer et for et. De semigrafiske tegn udskrives vha chr$ ( ), hvor parameteren i ( ) skal være tegnets nummer. De semigrafiske tegn er nummereret fra 128 og opefter ud fra princippet, at for hvert felt, der er sort skal der til 128 lægges den tilsvarende værdi:

1	2
4	8
16	32

F.eks. tegnet

har værdien 128 + 1 + 4 + 32 = 165

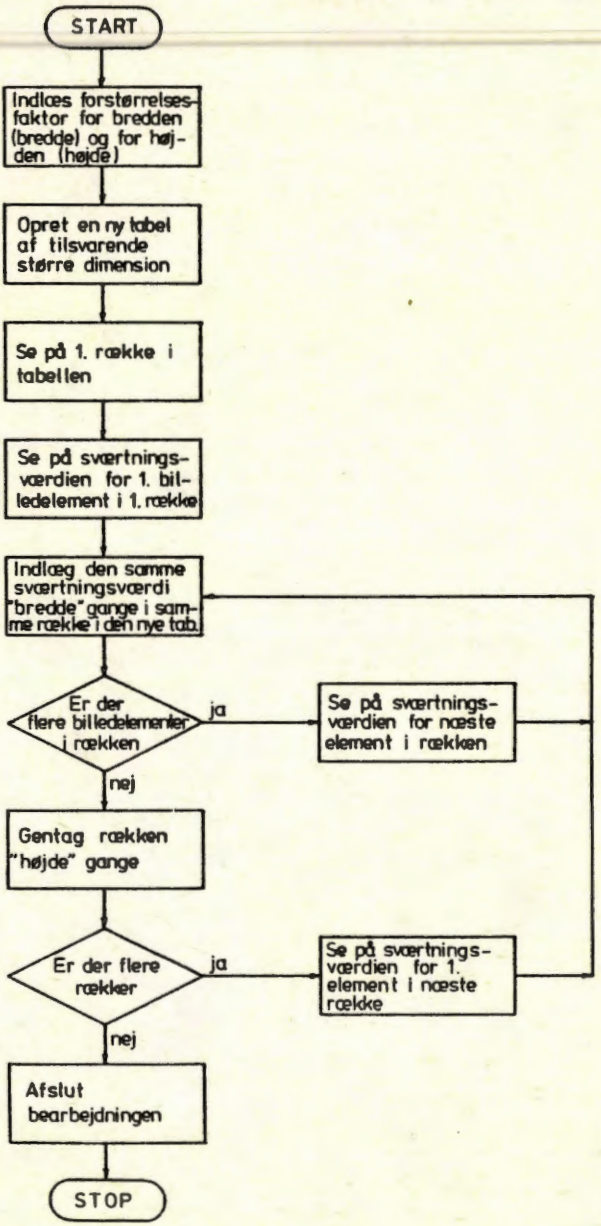
Vi vil ikke her se nærmere på selve udskrivningsprogrammet. Ligeledes har vi et indlæsningsprogram, vha hvilket vi kan indlæse og lagre vores billede i et array.

Vi har ved udarbejdelsen af algoritmen valgt kun at kunne forstørre med heltallige faktorer. Til gengæld kan vi forstørre med forskellige forstørrelsesfaktorer i bredden og i højden. Algoritmen er derfor lidt simplere end den foregående, principielt er de ens (med gennemløb række for række og herindenfor element for element).

Forstørrelsesalgoritmen til Cometen: Er gengivet på næste side.

Programering i Comal - 80

Udfra denne algoritme kan Comal - 80 programmet relativt nemt konstrueres. Jeg har programmet liggende, men mener ikke, at det er formålstjentligt at gengive det her, da det er uden nogen form for kommentarer og derfor svært - læseligt.


START
Indlæs forstørrelses-faktor for bredden (bredde) og for højden (højde)
Opret en ny tabel af tilsvarende større dimension
Se på 1. række i tabellen
Se på sværtnings-værdien for 1. billedelement i 1. række
Indlæg den samme sværtningsværdi "bredde" gange i samme række i den nye tab.
Er der flere billedelementer i rækken
ja
Se på sværtnings-værdien for næste element i rækken
nej
Gentag rækken "højde" gange
Er der flere rækker
ja
Se på sværtnings-værdien for 1. element i næste række
nej
Afslut bearbejdningen
STOP

Assemblerprogrammering på Comet'en

Ved John Plate

Assemblerprogrammering (eller som det også kaldes: programmering i symbolsk maskinsprog) er blevet stadig mere anvendt på tekniske skoler fordi visse opgaver nødvendiggør udnyttelse af faciliteter i mikrodatamater, som ikke umiddelbart er tilgængelige fra værtssproget, f.eks. Comal-80.
Der kan være tale om egentlig processtyring eller dataopsamling fra apparater eller maskiner som anvendes i undervisningen. Der kan nævnes små eksperimental-robotter, laboratorieudstyr eller værktøjsmaskiner mv.
De højere programmeringssprog som Comal-80 har gjort programmering til en rimeligt enkel aktivitet. De er konstrueret for at undgå kendskabet til adresser, ordlængder og alt det andet maskinnære. Det kan på en måde synes besynderligt at interessen for at komme i forbindelse med de vanskeligheder, der gemmer sig på dette niveau, er så stor. Den eneste begrundelse må være behovet for udnyttelse af specielle muligheder udenfor f.eks. Comal-80.
Men, hvis man nødvendigvis skal ned på maskinsprogsniveauet, bør man overveje de faciliteter som tilbydes i Pascal-dialekten Compas. Mange maskinnære opgaver kan løses hurtigt og elegant med Compas. Problemet med Compas er, at det ofte også fjerner eleverne fra at få en fornemmelse af. hvordan programmet fungerer. Det kan derfor være nødvendigt at benytte egentlig assemblerprogrammering i kombination med Comal. Så kan eleverne da forstå de væsentlige principper i Comal-programmet.
Der findes mange muligheder, hvoraf nogle almindeligt benyttede nævnes her.

1) Operationskoderne (svarende til assemblerinstruktionerne) kan lagres på decimal form i datasætninger og af Comal-programmet lagres i en datastruktur, f.eks. en tekst.

Principielt kan programmet bygges op således:

```
...
DIM asm$ OF n
FOR i:=0 TO n-1 DO
READ kode
POOK VARPTR(asm$)+i, kode
NEXT i
CALL VARPTR(asm$)
...
DATA (...)
```

Ulempen ved denne metode er, at det er en ret besværlig sag at skrive lidt større assemblerrutiner. Den manuelle korrekturlæsning og omformning til/fra decimal/hexadecimal repræsentation bliver utålelig og svær at spore fejl i. Desuden bliver programmer heller ikke pæne af den slags snørklerier.

2) En variant af ovenstående metode er at generere en CP/M-variant der kun disponerer over f.eks. 58 kb internt lager. De 2 kb frigjort lager kan anvendes til at lagre operationskoderne (assemblerinstruktionerne). På denne måde er det muligt at skrive mere komplekse rutiner, blandt andet fordi adresserummet på forhånd er kendt. Ulempen er at der skabes en CP/M-version som ikke ligner de andre man har i brug. Der kan let skabes forvirring på området. De fleste med lidt bitre erfaringer bag sig, ved at det ikke er en ønskelig situation.

3) Det er også muligt at reservere noget lager i Comal's dataområde. Ved DIMensioneringen af tekst skabes et område, hvor assemblerrutinen kan ligge, på samme måde som i eksempel 1).
Ved nogle forsøg kan man finde ud af, hvor Comal har lagt teksten og så skrive sin assemblerrutine udfra denne viden (ORG direktivet).
Fordelen er at man kan skrive sin assemblerrutine separat og foretage ændringer i Comal-programmet uafhængigt af assemblerrutinen.
Ulempen er at forandringer i Comal-programmet kan ændre det sted i lageret, hvor Comal har lagt den tekst, assemblerrutinen skal ligge i. Hvis det sker kan et kald af rutinen bevirke, at systemet løber løbsk.

4) Den fjerde mulighed (blandt mange andre tænkelige) er at assemblerrutinen konstrueres til at køre, ligegyldigt hvor i lageret den ligger. Eller rettere: Den finder selv ud af, hvor den ligger.
På samme måde som i eksempel 3) skrives Comal-programmet og assemblerrutinen hver for sig. Når Comal-programmet kører, indlæses assemblerrutinen fra en fil til en DIMensioneret tekstvariabel. Assemblerrutinen kaldes som sædvanlig med CALL sætningen.
Fordelen ved denne metode er, at Comal-programmet kan ændres helt frit, f.eks. af elever. Kun en lille PROC til indlæsning af assemblerrutinen skal kaldes før rutinen kan udføres fra programmet. Program og rutine kan skrives hver for sig og assemblerrutinen kan indkøres selvstændigt med et sporingsprogram (debugger).
Ulempen er at der kun kan anvendes Z80 assembler og at programmeringen bliver noget mere tricky. Til gengæld fås altså muligheden for at gøre anvendelsen af assemblerrutiner fuldstændig transparent for Comal-programmøren, f.eks. elever.

Jeg har udfærdiget en vejledning i assemblerprogrammering under Comal på Comet mikrodatamater, som især koncentrerer sig om eksempel 3) og 4). Den henvender sig til den lidt mere rutinerede assemblerprogrammør og viser (med eksempler) løsning på en række af de problemer man kan komme ud i.
Vejledningen foreligger i revideret udgave midt på sommeren og kan fås ved henvendelse. Vejledningen tænkes senere suppleret med eksempler på rutinebiblioteker, adressemodifikation og andre narrestreger.
Henvendelser bedes venligst bilagt en adresseret A4 svarkuvert.

Porte i mikrodatamater II

Ved Helge Jensen

Praktisk eksempel på anvendelse af porte

Portbegrebet dækker over mere end 'det sted hvor data bringes til / fra den ydre verden'. Portens anvendes også som hjælpefunktioner for at sikre datatransport. Det indebærer på mange måder fordele, at kombinere flere porte om e'n fælles opgave.

I det følgende gennemgås et eksempel på hvordan en port anvendes som parallel printerudgang og hvilken software der kræves for at kommunikere med en printer.

For at lette beskrivelsen vil eksemplet være baseret på parallel-stikket, som er placeret på bagsiden af en MPS-3000. Stikket er et 25 polet Cannon-stik, hvis ben er forbundet som et 'Centronic-snit'. Denne type forbindelse anvendes af et stort antal printere og andre ydre anheder, som skal kommunikere med en datamaskine.

Hvis du ser i hardware brugermanualen for den pågældende maskine, kan du finde adresserne for de benyttede porte, normalt adresse 240 og 241.

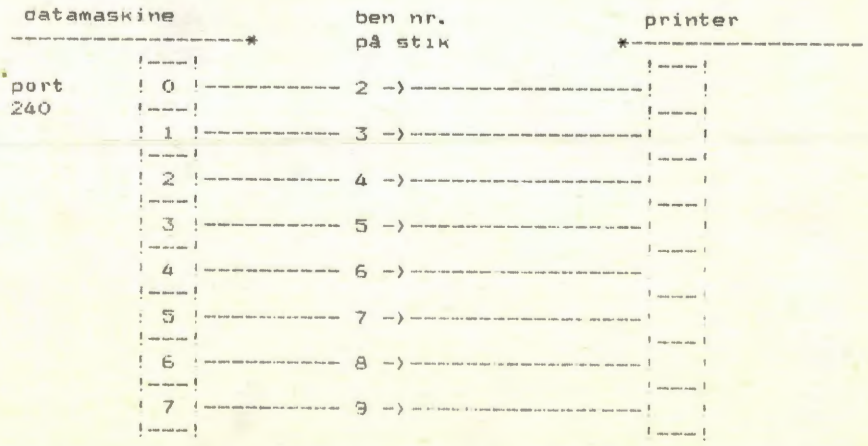
Problemet kan deles i to afsnit datamaskinen og printeren, hvor datamaskinen skal afgive og printeren modtage informationerne.

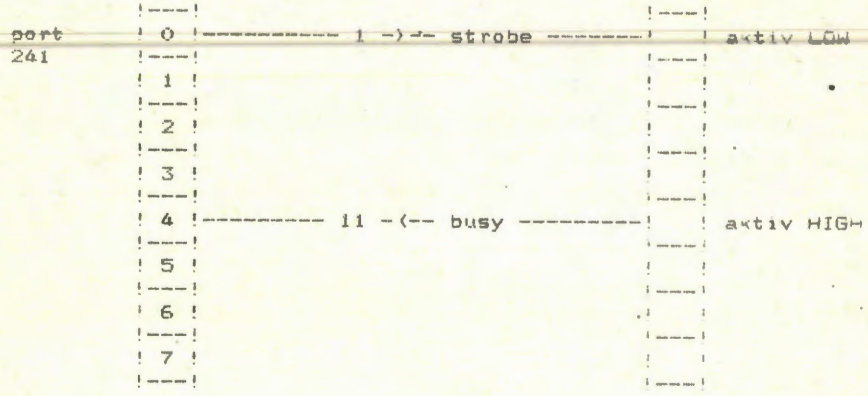
Datamaskinen skal anvende en port som udgang og printeren en port som indgang, hvor data kan transporteres igennem. Idet data fylder 8 bit kræver det 'en hel port som "landevej for data", der i eksemplet har adresse 240 og er initieret som udgang.

Problemet er nu, hvad der skal sikre og indikere at data er kommet korrekt frem til printeren, idet arbejdshastigheden for datamaskinen og printer er forskellige. Der findes her et signal fra datamaskinen som fortæller hvornår data er parrat til overføring og kan låses ind i printeren. Dette signal kaldes for 'strobe', det er en tidsmæssig indikation for valide data.

Et andet vigtigt problem er om printeren er optaget af andet arbejde og overhovedet 'lytter til data', til at varetage dette findes der et signal fra printer som benævnes 'busy'. Det er nødvendigt fordi der er mange andre ting end 'lytte efter data' printeren kan foretage sig, bla. betjene papirfremføring, vognretur, printe osv.

Model af opstilling:





Et helt forløb for overføring af data ser ud som følger:

- datamaskine ser efter busy
- data placeres i port 240
- strobe pulseres (går fra 1 til 0 tilbage til 1)

Data er nu placeret i printerens buffer og kan her overtages af printerens interne behandling, hvis det er en karakter af ASCII koden, vil den først fremkomme på papiret, når der sendes et linieskift eller en vognretur.

Et eksempel på software der printer 'et 'd' og derefter skifter linie. Programmet er skrevet i Comal-80.

```
0010 DIM B$ OF 8
0015 //
0020 DATAREGISTER:=240
0040 STATUSREGISTER:=241
0055 //
0085 BITNUMMER:=4
0090 STROBE_ON:=255
0100 STROBE_OFF:=254
0110 CR:=13
0111 LF:=10
0112 data:=ord("d")
0120 ////////////////////////////////////////////////////
0130 //  procedurer til at se om printeren er optaget  //
0140 //  samt give besked om at data er valide         //
0145 ////////////////////////////////////////////////////
0146 //

 *   Her skal portene initieres hvis der benyttes en anden
 *   end den benyttede PARALLEL-port.

0150 PROC BUSY
0160   REPEAT
0170     B$:=BSTR$(INP(STATUSREGISTER))
0180   UNTIL IVAL(B$(8-bitnummer))
0190 ENDPROC BUSY
0200 //
0210 PROC STROBE
0220   OUT STATUSREGISTER, STROBE_OFF
0230   OUT STATUSREGISTER, STROBE_ON
0240 ENDPROC STROBE
0245 ////////////////////////////////////////////////////
0250 //******** BRINGER EN KARAKTER TIL PRINTER ********
0280   EXEC BUSY
0310   OUT DATAREGISTER, data
0320   EXEC STROBE
0325 //******** GIVER ORDER OM VOGNRETUR ***************
0330   EXEC BUSY
0351   OUT DATAREGISTER, CR
0352   EXEC STROBE
0353 //******** GIVER ORDR OM LINIESKIFT ***************
0354   EXEC BUSY
0355   OUT DATAREGISTER, LF
0356   EXEC STROBE
```

Samme eksempel skrevet i COMPAS-PASCAL.

```
PROGRAM printer_eks;

CONST
    dataregister     = 240;
    statusregister   = 241;

    strobe_on        = 255;
    strobe_off       = 254;

    cr               =  13;
    lf               =  10;
    data             = 'd';


PROCEDURE busy;
BEGIN
  WHILE NOT(PORT(.statusregister.) AND 16 = 16 ) DO;
END;

PROCEDURE strobe;
BEGIN
  PORT(.statusregister.) := strobe_off;
  PORT(.statusregister.) := strobe_on;
END;


BEGIN

 *   Her skal portene initieres hvis der benyttes en anden
 *   end den benyttede PARALLEL-port.


  busy;
  PORT(.dataregister.) := ord(data);
  strobe;

  busy;
  PORT(.dataregister.) := cr;
  strobe;

  busy;
  PORT(.dataregister.) := lf;
  strbbe;

END.
```

Samme eksempel skrevet i ASSEMBLER-KODE:

```
                ORG     0100h
                ;.....................
buffer          EQU     240
contr           EQU     241
                ;.....................
busy            EQU     0001$0000b
strobe          EQU     1111$1110b
hvil            EQU     1111$1111b
                ;.....................
cr              EQU     13
lf              EQU     10
data            EQU     'a'
                ;.....................

            ;******** bringer en karakter til printer ********

*   Her skal portene initieres hvis der benyttes en anden
*   end den benyttede PARALLEL-port.

        start:
                CALL    busyx
                MVI     a,data
                OUT     buffer
                CALL    strobx
            ;******** giver order om vognretur
                CALL    busyx
                MVI     a,cr
                OUT     buffer
                CALL    strobx
            ;******** giver ordre om linieskift
                CALL    busyx
                MVI     a,lf
                OUT     buffer
                CALL    strobx
             ;******** gå tilbage til CP/M operativsystem
                CALL    0000
                ;
             ;*******************************************;
             ; subrutiner til at se om printeren er optaget;
             ; samt give besked om data er valide          ;
        strobx:
                MVI     a,strobe
                OUT     contr
                MVI     a,hvil
                OUT     contr
                RET
                ;
        busyx:
                IN      contr
                ANI     busy
                CPI     busy
                JNZ     busyx
                RET
                ;
                END
```

Det viste eksempel giver kun mulighed for at printe 'en karakter og går ud fra at portene er initieret, hvilket kun gælder sålænge PARALLEL stikket benyttes. Hvis du ønsker en anden port til kommunikationsudgang har du initieringen af portfunktionerne.

Det kræver ikke meget at initiere en port som udgang, der skal kun placeres decimalværdien 15 i kontrolregistret, hvorimod en port hvor der er blandede ind - og udgange kræves en ny metode for initiering.

For at kunne blande ind og udgange skal der først placeres decimalværdien 207 i kontrolregistret og derefter det ønskede bitmønster på samme adresse, hvor binærværdien i er udgang og 0 er indgang.

Lad os antage at bit nr. 7 og 0 i port adresse nr. 0 skal være udgang og resten indgange.

Eksempel skrevet i Comal-80:

```
0005 kontrolregister:=2
0010 bitmønster:=128+0+0+0+0+0+0+1
0020 OUT kontrolregister,207
0030 OUT kontrolregister,bitmønster
       //
```

Opgave:
Fremstil en printerdriver (procedure), som kan printe en vilkårlig streg med max. 80 karakterer og skifte linie, men kun hvis der gives besked med et flag ved navn linie. Afprøv rutinen på den eksisterende printerudgang og omskriv programmet så en plotter med 'Centronic-snit' kan tilsluttes en ny port som ligger på adresse 0 og 1. Adresse 0 er 8 bit parallel udgang og adresse 1 er mix-ind/ud. Rutinerne som er beskrevet kan fås ved at sende formateret diskette til forfatteren eller DaTS.

Løsningseksempel på opgaverne i blad nr.1 :

```
        (8)-7-6-5-4-3-2-1-0

32  =       0 0 1 0 0 0 0 0  = 32+0                      = 32
22  =       0 0 0 1 0 1 1 0  = 16+4+2                    = 22
35  =       0 0 1 0 0 0 1 1  = 32+2+1                    = 35
156 =       1 0 0 1 1 1 0 0. = 128+16+8+4                = 156
255 =       1 1 1 1 1 1 1 1  = 128+64+32+16+8+4+2+1      = 255
256 = (1)   0 0 0 0 0 0 0 0  = 256+0                     = 256
```

Løsningen skrevet i COMAL-80 :

```
0010 DATAREGISTER:=0       // GRUNDADRESSE
0020 KONTROLREGISTER:=0+2 // GRUNDADRESSE + 2
0021 FUNKTION:=15          // UDGANGSPORT
0022 motor1   :=1
0023 motor2   :=2
0024 lampe1   :=32
0025 lampe2   :=64
0042 //*****************************************
0045 // Programmet initierer PORT-A som udgang og
0046 // giver løsningen på opgaven
0047 //-----------------------------------------
0049 //
0050 OUT KONTROLREGISTER,FUNKTION
0060 OUT DATAREGISTER, INP(DATAREGISTER)+motor1+lampe1
0070 OUT DATAREGISTER, INP(DATAREGISTER)+motor2+lampe2
```

Samme løsning i assembler :

```
start:  ORG 0100H
;
datareg EQU 0000H
kontreg EQU 0000H + 2
funk    EQU 15
motor1  EQU 1
motor2  EQU 2
lampe1  EQU 32
lampe2  EQU 64
;
START:  MVI A,funk      ;Initier port-A
        OUT kontreg     ;som utgang
;
        IN  datareg
        ORI motor1+lampe1
        OUT datareg
;
        IN datareg
        ORI motor2+lampe2
        OUT datareg
;
        END
```

Samme løsning i COMPAS PASCAL :

```
PROGRAM port_eksempel;
const
kontrolregister = 2;
dataregister    = 0;
funktion        = 15;
motor1          = 1;
motor2          = 2;
lampe1          = 32;
lampe2          = 64;

BEGIN
  PORT(.kontrolregister.):=funktion;
  PORT(.dataregister.):=PORT(.dataregister.)+mo-
  tor1+lampe1;
  PORT(.dataregister.):=PORT(.dataregister.)+mo-
tor2+lampe2;
END.
```

Kombinatoriske Algoritmer II

Ved Ole Karmark

I sidste afsnit beskæftigede vi os med problemerne med generering af samtlige delmængder af en mængde som blev vist ved programmet DELMÆNG, desuden beskæftigede vi os med problemerne vedrørende permutation af en mængde. Dette blev vist med programmet PERMUTAT.

I denne artikel skal der først vises en konkret og praktisk anvendelse af en kombinatorisk algoritme. Den valgte problemstilling er at beskrive en algoritme, der genererer samtlige tipsrækker, et program,der kan generere tipsrækker, er listet. Dernæst diskuteres spørgsmålet om tidsfaktoren og spørgsmålet om evalueringsbetingelser. Herefter behandler artiklen problemet med at generere permutationer med gentagelse. En procedure permgen angives.

Generering af tipsrækker

I forbindelse med anvendelsen af datamater er en jævnlig tilbagevendende forespørgelse, om ikke man kunne betjene sig af datamaten ved beregning og udskrivning af tipsrækker. Såfremt man har sådanne tanker og overvejelser om hvorledes datamaten kan udnyttes til at finde tipsrækker, der set ud fra "tipperens" synspunkt er interessante, har man behov for en metode, hvormed tipsrækkerne kan genereres systematisk.
Opgaven kan formuleres således:

> Beskriv en fremgangsmåde,
> der muliggør generering af
> samtlige tipsrækker for en
> tipskupon med tretten kampe.

Et af de spørgsmål, der kan drøftes, er, hvor mange tipsrækker er der ialt og hvorledes kan dette tal beregnes. Beregningen af antallet af tipsrækkerne kan foregå på følgende måde:

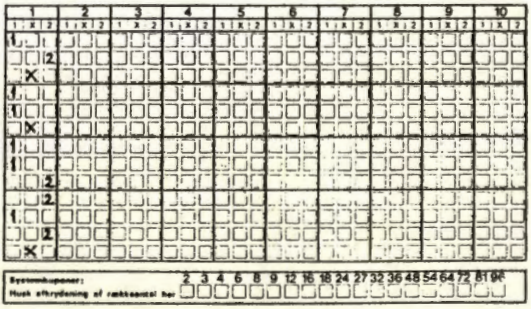
For hver kamp er der tre tegnmuligheder: (1,X,2), dvs. at der er tre mulige udfald for hver kamp. Da der ialt er tretten kampe ses det, at det søgte tal frembringes af følgende sammensatte valgproces:

1. kamp: 1. tegn vælges (3 muligheder)
2. kamp: 2. tegn vælges (3 muligheder)
3. kamp: 3. tegn vælges (3 muligheder)
-
-
13. kamp: 13 tegn vælges (3 muligheder)

Antallet af forløb for valgprocessen og dermed antallet af tipsrækker er:

$$\underbrace{3 \cdot 3 \cdot 3 \text{ ----- } \cdot 3}_{\text{Ialt tretten tretaller}} = 3^{13} = 1.594.320$$

Nedenfor er vist en tipskupon, som den fremstilles af tipstjenesten:



Fremover vil vi angive tipsrækker, som den på kuponen viste, på følgende måde:

Tipsrække: | 1 | 2 | X | 1 | 1 | X | 1 | 1 | 2 | 2 | 1 | 2 | X |

Når vi skal beskrive en algoritme til generering af tipsrækkerne, kan vi sammenligne denne proces, med en proces, der går ud på at skrive samtlige 13-cifrede tal, når vi kun må anvende 3 tal nemlig: 0,1,2,. Den proces kan anskueliggøres ved følgende opstilling:

Ciffer nr. 1 2 3 4 5 6 7 8 9 10 11 12 13

For hvert af de tretten cifre har vi tre tal til rådighed (0,1,2). Det skulle således fremgå, at det er en opgave af samme art som bestemmelse af antallet af tipsrækker. Det mindste tal vi kan skrive er:

Cifferrække nr. 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Som numerisk værdi betragtet repræsenterer cifferrækken tallet nul. Det væsentligste for os er imidlertid, at vi har valgt tallet 0 for hvert ciffer. Med dette tal (cifferrækken nr. 1) som udgangstal kan vi nu systematisk skrive resten, idet vi blot for hvert nyt tal gør det én større end det foregående (numerisk set).

Nedenfor er vist nogle få af de mulige genereringer:

Cifferrække nr. 1: 0 0 0 0 0 0 0 0 0 0 0 0 0

Cifferrække nr. 2: 0 0 0 0 0 0 0 0 0 0 0 0 1

Cifferrække nr. 3: 0 0 0 0 0 0 0 0 0 0 0 0 2

(mente overførsel)

Cifferrække nr. 4: 0 0 0 0 0 0 0 0 0 0 0 1 0

Cifferrække nr. 5: 0 0 0 0 0 0 0 0 0 0 0 1 1

Cifferrække nr. 6: 0 0 0 0 0 0 0 0 0 0 0 1 2
(mente overførsel)

Cifferrække nr. 7: 0 0 0 0 0 0 0 0 0 0 0 2 0

Cifferrække nr. 8: 0 0 0 0 0 0 0 0 0 0 0 2 1

Cifferrække nr. 9: 0 0 0 0 0 0 0 0 0 0 0 2 2
(mente overførsel)

Cifferrække nr. 10: 0 0 0 0 0 0 0 0 0 0 1 0 0

Cifferrække nr. 11: 0 0 0 0 0 0 0 0 0 0 1 0 1

-

-

osv.

Den ovenfor skitserede proces kan også opfattes som angivelse af tallene fra 0 til 1.594.319 i 3-talsystemet. At den beskrevne metode kan udnyttes ved fremstilling af tipsrækker ses på den måde,at vi ved udskrivningen af rækkerne blot skal fortolke tallet 0 som en repræsentant for et X (et kryds), og dermed fås følgende fortolkning af den ovenfor viste generering:

Række nr. 1: X X X X X X X X X X X X X

Række nr. 2: X X X X X X X X X X X X 1

Række nr. 3: X X X X X X X X X X X X 2

Række nr. 4: X X X X X X X X X X X 1 X

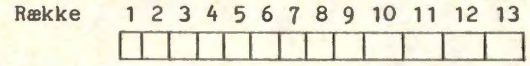
Række nr. 5: X X X X X X X X X X X 1 1

-

-

osv.

På baggrund af den ovenfor givne fremgangsmåde kan vi nu udvikle og beskrive en algoritme til udskrivning af tipsrækkerne. Ved beskrivelsen af algoritmen anvender vi talsættet Række bestående af tretten elementer:




```
Algoritme tipsræk

Opret talsættet Række (13)

Sæt rækkeelementerne til nul

Sæt flere til sand

Sålænge [ flere ]

Udfør
    Udskriv Række

    For i: = 13 nedtil 1

    Udfør hvis Række (i)< 2

          så   Række (i): = Række (i)+1

               For j: = i+1 optil 13

               Udfør [ Række (j): = 0 ]

               gå til ud

    Sæt flere til falsk

    Ud:
```

Det bemærkes, at når vi foretager en menteoverførsel skal alle elementerne til højre for det element nr,, der er blevet forøget med én, nulstilles - som gælder ved almindelig sammenlægning af tal. Algoritmen slutter - som det fremgår- når ingen af elementerne kan forøges eller sagt på en anden måde, når alle elementerne er sat til 2.

Program listning.

```
0010 //PROGRAM TIL UDSKRIVNING AF TIPSRÆKKER.
0020 //
0030 PROC UDSKRIV
0040  TÆLLER:=TÆLLER+1
0050  PRINT USING "Tipsrække nr. ######## ": TÆLLER,
0060  FOR K:=1 TO 13 DO
0070   IF RÆKKE(K)=0 THEN
0080    PRINT "X",
0090   ELSE
0100    PRINT RÆKKE(K),
0110   ENDIF
0120  NEXT K
0130  PRINT
0140 ENDPROC UDSKRIV
0150 //
0160 DIM RÆKKE(13)
0170 //
0180 //     ****  HOVEDPROGRAM  ***
0190 MAT RÆKKE:=0
0200 TÆLLER:=0
0210 FLERE:=TRUE
0220 WHILE FLERE DO
0230  EXEC UDSKRIV
0240  FOR I:=13 DOWNTO 1 DO
0250   IF RÆKKE(I)<2 THEN
0260    RÆKKE(I):=RÆKKE(I)+1
0270    FOR J:=I+1 TO 13 DO
0280     RÆKKE(J):=0
0290    NEXT J
0300    GOTO UD
0310   ENDIF
0320  NEXT I
0330  FLERE:=FALSE
0340 LABEL UD
0350 ENDWHILE
0360 //      ***  SLUT  ***
0370 END
```

Eksempel på udskrift fra program Tipsræk:

```
Tipsrække nr.     1 XXXXXXXXXXXXX
Tipsrække nr.     2 XXXXXXXXXXXX1
Tipsrække nr.     3 XXXXXXXXXXXX2
Tipsrække nr.     4 XXXXXXXXXXX1X
Tipsrække nr.     5 XXXXXXXXXXX11
Tipsrække nr.     6 XXXXXXXXXXX12
Tipsrække nr.     7 XXXXXXXXXXX2X
Tipsrække nr.     8 XXXXXXXXXXX21
Tipsrække nr.     9 XXXXXXXXXXX22
Tipsrække nr.    10 XXXXXXXXXX1XX
Tipsrække nr.    11 XXXXXXXXXX1X1
Tipsrække nr.    12 XXXXXXXXXX1X2
Tipsrække nr.    13 XXXXXXXXXX11X
Tipsrække nr.    14 XXXXXXXXXX111
Tipsrække nr.    15 XXXXXXXXXX112
Tipsrække nr.    16 XXXXXXXXXX12X
Tipsrække nr.    17 XXXXXXXXXX121
Tipsrække nr.    18 XXXXXXXXXX122
Tipsrække nr.    19 XXXXXXXXXX2XX
Tipsrække nr.    20 XXXXXXXXXX2X1
Tipsrække nr.    21 XXXXXXXXXX2X2
Tipsrække nr.    22 XXXXXXXXXX21X
Tipsrække nr.    23 XXXXXXXXXX211
Tipsrække nr.    24 XXXXXXXXXX212
Tipsrække nr.    25 XXXXXXXXXX22X
Tipsrække nr.    26 XXXXXXXXXX221
Tipsrække nr.    27 XXXXXXXXXX222
Tipsrække nr.    28 XXXXXXXXX1XXX
Tipsrække nr.    29 XXXXXXXXX1XX1
Tipsrække nr.    30 XXXXXXXXX1XX2
Tipsrække nr.    31 XXXXXXXXX1X1X
Tipsrække nr.    32 XXXXXXXXX1X11
```

Et af problemerne ved anvendelsen af program Tipsræk er, at det tager meget lang tid at få genereret samtlige rækker ca. 12 timer

Det står klart at selve udskrivningen sammenlagt beslaglægger en stor del af tiden. For at gøre programmet mere effektivt kunne man undlade proceduren Udskriv. Hvem ville iøvrigt være interesseret i at se samtlige rækker? Ingen - forhåbentligt. Det interessen centrerer sig om, er at udskrive rækker med særlige karakteristika, rækker der opfylder bestemte betingelser, betingelser for hvor mange ettaller, krydser og tabeller, betingelser for hvor mange ettaller på stribe, krydser på stribe, osv. Disse betingelser kan opstilles på baggrund af erfaringer om, hvad der særligt kendetegner de "tretten rigtige". Man kan med et sådant formål for øje formulere en evalueringsfunktion, der for hver ny række undersøger om rækken opfylder de stillede betingelser. Såfremt rækken opfylder betingelserne udskrives den ellers afvises den, og en ny række genereres. På denne måde kan man opnå at kun rækker, der har interesse udskrives.

Spørgsmålet om hvor lang tid genereringen tager spiller stadig en rolle. Vi kan gøre programmet mere effektiv og vi kan undlade udskrivningen, stadigvæk tager det temmelig lang tid. Benyttes komplicerede evalueringeskriterier er vi lige vidt, hvad angår tidsfaktoren. For at gøre problemløsningen (udskrivningen af rækker, der opfylder betingelserne) hurtigere, er der nu flere muligheder. Man kunne vælge at formulere programmet i et andet programmeringssprog f.eks. pascal, eller mankan skrive de tidskrævende rutiner i assemblersprog. En anden mulighed er at beskrive algoritmen udfra de opstillede betingelser, derved kan det -opnås at kun en mindre del af mængden af tipsrækkerne genereres og ud af denne delmængde kan de ønskede rækker findes.

Læseren opfordres til at forsøge sig med en eller flere af de skitserede muligheder. Man kan forsøge sig med følgende problem:

Udskriv samtlige tipsrækker, der opfylder betingelserne:

1. Antallet af ettaller er fem
   Antallet af krydser er fire
2. Kamp nr. 1 er et ettal
   Kamp nr. 12 er et kryds
3. Der er netop 3 krydser på stribe
   Der er netop 3 totaller på stribe

Generering af permutationer med gentagelse

På baggrund af den opstillede algoritme til fremstilling af samtlige tipsrækker, kan vi , hvis vi studerer eksemplet nøjere udlede en generel regel for generering af Permutationer med gentagelse. Lad os antage, at vi skal foretage en permutering med gentagelse af cifrene: 1,2,3,---,n. Vi begynder med rækken: 111---1. Det sidste ciffer forøges med én indtil rækken: 111---1n er fremstillet. Herfra er det ikke længere muligt at øge det sidste ciffer, og vi tvinges da til at lægge én til det (n-1)'te ciffer. Dernæst sætter vi det n'te ciffer til 1, og vi har hermed genereret rækken: 111---21.

Denne metode kan generaliseres på følgende vis: ud fra en mængde bestående af n elementer skal der dannes permutationer med gentagelse bestående af r elementer. Denne generering skal foretages udfra den foregående permutation. Vi betragter nu r-permutationen: $m_1m_2m_3---m_r$. En algoritme, der giver næste permutation er følgende:

1. Find det største element i således at $m_i < n$
2. Læg én til m_i
3. Sæt $m_{i+1} = m_{i+2} = m_{i+3} = \ldots = m_r = 1$

Vi bemærker, at hvis i er lig med r, som det vil være i mange tilfælde, så udfører det 3. trin i algoritmen ingenting.

En procedure for generering af r-permutationer med gentagelse kan se således ud:

```
0010 PROC PERMGEN(N, R)
0020  IF FØRSTE THEN
0030   FOR I:=1 TO R DO
0040    RÆKKE(I):=1
0050   NEXT I
0060   FØRSTE:=FALSE
0070   FLERE:=TRUE
0080   GOTO UD
0090  ENDIF
0100  //
0110  FOR I:=R DOWNTO 1 DO
0120   IF RÆKKE(I)<N THEN
0130    RÆKKE(I):=RÆKKE(I)+1
0140    FOR J:=I+1 TO R DO
0150     RÆKKE(J):=1
0160    NEXT J
0170    GOTO UD
0180   ENDIF
0190  NEXT I
0200  FLERE:=FALSE
0210 LABEL UD
0220 ENDPROC PERMGEN
```

Den viste procedure kan anvendes på lignende måde som vist med proceduren Perm i program Permutat fra den tidligere artikel. Nedenfor er vist et eksempel på en kørsel med teksten "1X2" som inputdata (n = 3), r er sat til 13.

```
  1: 111111111111    2: 11111111111X    3: 111111111112    4: 11111111111X1
  5: 1111111111XX    6: 11111111111X2    7: 111111111121    8: 11111111112X
  9: 111111111122   10: 11111111X11   11: 11111111X1X   12: 11111111X12
 13: 11111111XX1   14: 11111111XXX   15: 11111111XX2   16: 11111111X21
 17: 11111111X2X   18: 11111111X22   19: 1111111111211   20: 111111111121X
 21: 11111111212   22: 11111111X2X1   23: 1111111112XX   24: 1111111112X2
 25: 111111111221   26: 11111111122X   27: 111111111222   28: 11111111X111
 29: 11111111X11X   30: 11111111X112   31: 11111111X1X1   32: 11111111X1XX
 33: 11111111X1X2   34: 11111111X121   35: 11111111X12X   36: 11111111X122
 37: 1111111XX11   38: 11111111XX1X   39: 11111111XX12   40: 1111111XXX1
 41: 11111111XXXX   42: 11111111XXX2   43: 11111111XX21   44: 11111111XX2X
 45: 11111111XX22   46: 11111111X211   47: 11111111X21X   48: 11111111X212
 49: 11111111X2X1   50: 11111111X2XX   51: 11111111X2X2   52: 11111111X221
 53: 11111111X22X   54: 11111111X222   55: 1111111112111   56: 111111111211X
 57: 1111111112112   58: 111111111211X1   59: 11111111121XX   60: 11111111121X2
 61: 1111111112121   62: 11111111212X   63: 1111111112122   64: 111111112X11
 65: 111111112X1X   66: 111111112X12   67: 11111111 2XX1   68: 111111112XXX
 69: 111111112XX2   70: 111111112X21   71: 111111112X2X   72: 111111112X22
 73: 11111111122211   74: 111111111221X   75: 1111111112212   76: 11111111122X1
 77: 1111111122XX   78: 111111111122X2   79: 111111112221   80: 11111111222X
 81: 1111111112222   82: 11111111X1111   83: 11111111X111X   84: 11111111X1112
 85: 111111111X11X1   86: 11111111X11XX   87: 11111111X11X2   88: 11111111X1121
 89: 11111111X112X   90: 11111111X1122   91: 11111111X1X11   92: 11111111X1X1X
 93: 11111111X1X12   94: 11111111X1XX1   95: 11111111X1XXX   96: 11111111X1XX2
 97: 11111111X1X21   98: 11111111X1X2X   99: 11111111X1X22  100: 11111111X1211
101: 11111111X121X  102: 11111111X1212  103: 11111111X12X1  104: 11111111X12XX
105: 11111111X12X2  106: 11111111X1221  107: 11111111X122X  108: 11111111X1222
109: 11111111XX111  110: 11111111XX11X  111: 11111111XX112  112: 11111111XX1X1
113: 11111111XX1XX  114: 11111111XX1X2  115: 11111111XX121  116: 11111111XX12X
117: 11111111XX122  118: 11111111XXX11  119: 11111111XXX1X  120: 11111111XXX12
121: 11111111XXXX1  122: 11111111XXXXX  123: 11111111XXXX2  124: 11111111XXX21
```

Maskinen

Anmeldt af Erik Nordholdt

Tracy Kidder: "Maskinen"
328 sider.
(Nyt Nordisk forlag, Arnold Busck.)

"De stenhuggere, der rejste katedralerne, arbejdede formodentlig kun delvis for pengenes skyld. De byggede jo templer til Gud. Det var et af den slags arbejder, der gav livet mening. Jeg tror det var det, West og hans team søgte efter. De sagde ofte selv, at de ikke arbejdede på deres maskine for pengenes skyld."..."Mange søgte efter de rette ord for at kunne beskrive, hvad der var den egentlige løn af indsatsen. De brugte vendinger som "selvrealisation", "en følelse af at have præsteret noget" og "selvtilfredsstillelse"".

Gennem bogens godt 300 sider får vi en flammende beskrivelse af, hvordan et hold unge ingeniører og dataloger, "de hårde og de bløde drenge" fra det amerikanske firma "data General" i løbet af et par år fra foråret 1978 til foråret 1980 udviklede en 32 bit super-minidatamat.

Baggrunden var den, at firmaet DEC (Digital Equipment Corporation) havde præsenteret vax 11/780 i oktober 1977, en 32 bit minidatamat, der kunne adressere 4 mia tegn i stedet for kun 65000, som de sædvanlige 16 bit maskiner kunne klare. De øvrige edb-producenter, herunder også Data General måtte så af konkurrencemæssige grunde hurtigt svare igen med en lignende nyhed.
Forfatteren fulgte processen på nært hold gennem det meste af perioden. Der er ikke blot tale om en teknisk redegørelse for konstruktionsforløbet. Forfatteren giver en intens beskrivelse af, hvorledes projektet opstod, projektlederen Tom Wests kamp mod modstridende interesser i det store firma, og desuden holdets arbejde dag og nat. Forfatterens egentlige hensigt med bogen er tydeligvis at beskrive de enkelte medarbejderes følelse for arbejdet.

Vi får en - måske lidt for grundig og til tider kedelig gennemgang af hver enkelts barndom, uddannelse og syn på arbejdet. Det er i denne forbindelse påfaldende, at alle disse ingeniører og dataloger begyndte deres karriere ved som barn at "skille ure og radioer ad og sætte dem sammen igen". Men forfatteren har virkelig haft held med at klargøre for læseren, hvorledes medarbejdernes optagethed af maskinen kan holde dem fangen i kælderen 80 timer om ugen gennem hele perioden.

"Stil dem ned i mørket, prop dem med lort og se dem gro." Dette kaldes "champignon-ledelsesfilosofien" i Tracy Kidders bog "Maskinen".
Champignon sentensen er god, fordi den helt tydeligt gør udtryk for, at mennesket altså som noget grundlæggende skaber verden omkring sig. Trods mørke og lort vil mennesket skabe. - Og det er denne bogs inciterende styrke, at den ikke ser teknologien uden om mennesket, men netop ser den som et resultat af det legende menneske.

Moralen er imidlertid besk. Alle disse dygtige, kæmpende mennesker - undtagen lederen Tom West - bliver enten fyret eller forlader firmaet i depression over den manglende påskønnelse, da projektet er i hus. Det legende menneske er af værdi for tid og samfund. Det må bare her som alle vegne vige pladsen for kapitalens grådige og umættelige gab.

Hva datamaskinen ikke kan

Anmeldt af Kaj Thrysøe

Herman Ruge Jervell/Kaj A.Olsen: Hva datamaskinen ikke kan. Universitetsforlaget. Oslo 1982

Der er desværre langt imellem kritiske fremstillinger af, hvad datamaskiner kan/ikke kan fra EDB-specialisternes egen pen. De fleste fremstillinger om datamaskiner er præget af "Neanderthalteknologerne's" (udtrykket er "lånt" af Børge Christensen) ukuelige fremskridtstro. I følge den dominerende strømning er der (næsten) ingen grænser for, hvad datamaskiner kan. -Derfor er der grund til at glæde sig over, at to norske videnskabsmænd (en matematiker og en datalog) har skrevet en let tilgængelig bog, med det formål at beskrive hvad datamaskiner ikke kan.
Forfatterne har tydeligt været irriteret over "... de ingeniørromantiske visioner om at datamaskiner skulle kunne være oversættere, læge, lærer osv. . De fandt det nødvendigt at understrege at dette var en grov overvurdering af hvad datamaskiner kan". Forfatterne mener med rette, at sådanne romantiske visioner kun forvirrer debatten og skaber unødvendig ængstelse hos de mennesker, der kan blive berørt af EDB-teknologien.

Nu er det efterhånden 40 år siden de første datamaskiner blev fremstillet, og i den tid er der indhøstet en god del erfaring med hvad datamaskiner kan, og hvad de ikke kan. Et er sikkert: datamaskiner stiller klare krav til de områder de skal anvendes på.
Der kræves præcist et område, der er

- vel afgrænset,
- helt formaliseret og
- entydigt.

Disse grundbegreber for anvendelse af EDB-maskiner er bogens pointe, og samtidig "den røde tråd" i fremstillingen af hvor datamaskinen støder på sine grænser.

Bogen er bygget op i to dele med hvert et antal kapitler, der behandler centrale størrelser i forbindelse med EDB-maskinen.
Del 1 giver en indføring i EDB-maskinen, programmering og hvad datamaskiner bruges til. Denne indføring skal samtidig danne grundlag for fremstillingen og argumentationen i bogens anden del.
Del 2 består af otte kapitler, der behandler så vigtige emner som Sprog, Datamaskiner og Sprog, Den tænkende Maskine, Formalisering m.m..
Bogens vigtigste begrebspar er åbne- og lukkede systemer:

Følgende er åbent	Følgende er lukket
Verden	Regnskab
Sproget	Tegnmanipulationer
Meninger	Bevægelser
Handlinger	Skema'er
Mennesket	Datamaskiner.

Der er ganske vist en glidende overgang mellem åbne og lukkede systemer, men datamaskinen kræver et helt lukket system. Dette giver en grov principiel grænse for hvad datamaskiner kan. (s.58f)
Det naturlige sprog er åbent, fordi det skal bruges til at beskrive en kompleks virkelighed. Vores dagligdags sprog er meget varieret og indeholder mange tvetydigheder. Et varieret rummeligt sprog giver tilværelsen mere dybde og sætter os i stand til at formidle kvalitative oplevelser. Vores naturlige sprog tilhører heldigvis den åbne del af virkeligheden. -Hvis vi prøver at gøre sproget lukket, vil det ikke kunne have alle de funktioner et naturligt sprog bør have.
Det naturlige sprog's åbne karakter er baggrunden for, at EF's forsøg på automatisk at oversætte dokumenter fra et nationalsprog til et andet ikke har været særligt vellykkede. Bogen beskriver et mislykket forsøg på oversættelse fra russisk til engelse. Et par eksempler:

We demand peace.	translatøroversættelse
WE REQUIRE WORLD.	EDB-oversættelse
No more for me. thank you.	translatøroversættelse
I NO LONGER WANT THANKS.	EDB-oversættelse

Det er forståeligt, at translatørerne i Bruxelles ikke længere ser nogen trussel i deres erhverv fra EDB-maskinerne. (s.70f)
Det er klart, at EDB-maskiner vil overtage en del arbejde, som i dag udføres af mennesker. -Men fordi datamaskiner kun kan arbejde i lukkede systemer, vil det først og fremmest blive det rutineprægede arbejde, der vil blive automatiseret. Og dette er ikke altid sikkert. Man automatiserer ikke bare ud i den blå luft; det skal også kunne betale sig. Her overser debattørerne ofte at Soft-Warearbejdet er langsommeligt og dyrt. Faktisk stiger omsætningerne til programudvikling eksponentielt med antallet af programlinjer. Bogen giver følgende eksempel (s.146):

System	Totalpris (1981 n.kr.)	Antal Programlinje	Pris pr. linje
store (univ.system)	3o.ooo.ooo	1oo.ooo	3oo
middel (sundhedsarkiv)	3.ooo.ooo	3o.ooo	1oo
mindre (tekstbehandl.)	5oo.ooo	1o.ooo	5o

Ovennævnte eksempler skulle gerne skærpe interessen for at læse denne bog, der er velgørende i sin afmystificering af EDB-maskiner og disses muligheder. George Orwell's "1984" er ikke så tæt på, som nogle gerne vil bilde os ind.
Bogen er ikke en lærebog; men et indlæg i debatten om datamaskiner og hvad vi kan vente os fra den side i den nærmeste fremtid.

Adresser:

Kaj Thrysøe
Århus Tekniske Skole
Halmstadgade 5
8200 Århus N

John Plate
Laborantskolen i Roskilde
Maglegårdsvej 8
4000 Roskilde

Helge Jensen
Viborg Tekniske Skole
H.C. Andersensvej 7-9
8800 Viborg

Erik Nordholdt
Århus Tekniske Skole
Halmstadgade 6
8200 Århus N

Lone Verner Nielsen
Århus Tekniske Skole
Halmstadgade 6
8200 Århus N

Ole Karmark
Århus Tekniske Skole
Halmstadgade 6
8200 Århus N