

STACKPOINTER

Husorgan för STACKEN, datorföreningen på KTH.
Grundad 1978

1-82



64
sidor!

STACKPOINTER

är organ för datorföreningen **STACKEN**.

STACKPOINTER utkommer när material i tillräcklig mängd finnes, normalt 2 - 3 gånger per år.

Ansvarig utgivare:

Per Lindberg

Redacteur:

Dag Rende

I redaktionen:

Björn Carlsson

Robert Lindh

Per Danielsson

Hans Nordström

Ulf Landgren

Dag Rende

Per Lindberg

Jan Michael Rynning

Datorföreningen **STACKEN**

c/o NADA, KTH

100 44 STOCKHOLM

Postgiro: 433 01 15 - 9

Medlemskap i **STACKEN** kan beviljas efter ansökan till föreningens styrelse. Medlemskapet kostar f.n. 70 SEK per år och inbetalas på **STACKENS** postgirokonto.



INNEHÅLL

1	Framsida. Frontpanel.
2	Datorföreningen Stacken. Information om föreningen och dess organ.
3	Innehåll. Se denna sida.
4	Redaktörens ruta.
5	Ordföranden har ordet.
8	Lägesrapport, 16-bitgruppen. Prototyp på gång. Snart dax att vira.
9	Dagsläget för AMIS. Stackens stolthet såld till SIMPROG.
9	Studiebesök på EMS! Intresserad av elektronmusik? Stacken gör besök på elektronmusikstudion.
10	Problemhörnan. Roliga problem med programmeringsanknytning.
11	Kallelse till årsmöte. Årets största tilldragelse? Kom och rösta fram en ny styrelse, budget, och ev. nya stadgar!
12	Motioner till årsmötet. -Nya, och förhoppningsvis enklare stagar. -Utträde ur Unga Foscare. Belslutas på årsmötet!
14	Grafiska Grundrutiner. Idéer om hur ett bra grafikpaket kan byggas upp.
20	Textplottning. Transformera, translatera, skala, vrid och vänd.
22	Rekursiva kurvor. - Snygga att se på. Gör dem själv!

27	Streckkod till HP-41. Låt din kalkylator läsa STACKPOINTER!
28	Rekursiv cirkelritning. Avrundning av grafikartiklarna.
40	SOFTNET - data på radio. Ett amatörradionät för data-överföring håller på att konstrueras i Linköping.
43	IBM 370 op-koder. ...eller vad det nu är...
44	Partialevaluering. En metod som kan användas för att, till synes magiskt, framställa kompilatorer.
48	SAIL - Hackervänligt programmeringsspråk från Stanford.
51	NS16032. Nationals kommande mikroprocessor - en korsning mellan VAX och PASCAL MicroEngine.
54	68000. En närsynt titt på Motorolas lyxiga processor.
56	Lispmaskinen CADR. Per-sondatorn för finsmakare, programmerad helt i LISP.
60	Medlemslista. Matrikel för föreningen.
63	Hackern. Det blev visst en kultursida, i alla fall...
64	Baksidan. En illustration över hackandets baksida.

Redaktörens Ruta



STACKPOINTER har äntligen gått över till "riktig" datorsättning! I förra numret använde vi visserligen ett formatteringsprogram, SCRIBE, men skrev ut resultatet på en typhjulsskrivare, Diablo. Omöjligt att byta typsnitt här och var alltså. Nu använder vi TeX, Donald E. Knuths nya system. Utmatningen gör vi på en elektrostatisk skrivare, Versatec V-80.

Som du ser blir resultatet mer proffsigt jämfört med förra numret (du kan leta fram nummer 1-78 och jämföra Det också). Det bättre resultatet för man betala i form av mer arbete, man blir ju petigare med utseendet på artiklarna och måste använda **alla** finnesser.

- Så någon rationalisering har inte TeX inneburit.

Inte heller till detta nummer har det varit någon brist på manus, och redan ligger manus och väntar på nästa nummer. Goda tider alltså.

Medan BYTE mer och mer börjar likna telefonkatalogens yrkesregister, håller STACKPOINTER stilen.

Dag Rende

ORDFÖRANDEN HAR ORDET



INC SP

Har ni hört att företaget som rimmar på ni-vet-vem har börjat sälja sin stora mainframe integrerad på en chip, med minne och allt? För 500 dollar!? Nej, det är inte sant, tänker du, men man vet ju aldrig... innan det här numret av STACKPOINTER lämnat pressarna är världen ytterligare några dagar äldre, och alla vet ju hur fort utvecklingen går i datorvärlden! Som ordförande i en skara datorentusiaster blir man dubbelt stressad: dels ska man hålla sig ajour med alla framsteg som dyker upp runt en likt vitsippor i en skogsbacke om våren, dels ska man styra och ställa i en mycket vital och rolig förening, där det finns nästan hur mycket som helst att göra, och där allt händer på en gång. Nå, nu ska jag väl få mer tid över till det första, då jag slutar som ordförande i April. Men därmed inte sagt att jag försvinner fullständigt från STACKEN, det finns alldeles för många bra skäl att stanna kvar (och bli medlem, hint, hint!) i STACKEN! Och när vi kanske snart ska få tillgång till ett Wicat-system med en MC68000 och lite annat smått och gott i, är det onekligen svårt att slita sig! (Det är ITM som inköpt en dylik burk, och som lovat att STACKEN ska få leka med den, bara den är ledig).

RESUME: DETTA HAR HÄNT _____

Sedan STACKEN grundades 1978, har vi upplevt ungefär en generationsväxling, och ett ökat intresse för mjukvara. Hårdvaruentusiasterna är dock ingalunda avskaffade, utan lever och trivs, och har bl.a. skruvat samman en Motorola 68000 med lite annan osorterad bråte, och fått grejerna att funka! Det har även varit tal om en grafisk terminal, både på STACKEN, Lysator (vår systerorganisation i Linköping) och mikrodatorklubben på Svenska IBM. Den terminalbyggsats som gjorts på Lysator är klar sedan länge, och fungerar sedan lång tid tillbaka, vilket säkert några på STACKEN som var med i samköpet kan intyga.

Och på mjukvarusidan har det inte heller stått stilla, vill jag lova! STACKENS förnämsta (?) mjukvaruprojekt hittills, den EMACS-kompatibla editorn AMIS, har lyckats nästan för bra! Det kommersiella intresset blev så starkt, att STACKEN inte längre har möjlighet att distribuera den till alla som vill ha en portabel standardeditor på sin FOO/4711. Rättigheterna har därför sålts till ett nystartat företag som heter SIMPROG. Rättigheterna till den icke-kommersiella delen av den akademiska världen har dock STACKEN kvar, men överläter gärna distribution och underhåll på SIMPROG. AMIS används numera i undervisningen på NADA, och av alla andra som behöver en stark texteditor, och som vill ha samma kommandon till sin favoriteditor, vare sig hon kör AMIS under TOPS-10 på Nadja, eller EMACS under TOPS-20 på låt oss säga AIDA, eller MINCE under CP/M eller UNIX på sin egen mikrodator.

Men allt detta är ju historia, och därför ganska ointressant. Det finns ett berömt citat som blivit något av ett bevingat ord: "Jag är intresserad av framtiden, eftersom jag kommer att tillbringa resten av mitt liv där" –Så låt oss rikta blickarna in i framtiden, och se vad som händer på STACKEN där!



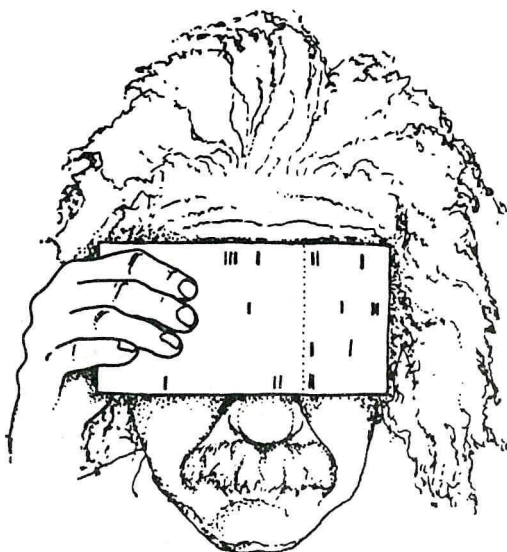
FUTURUM SPECULATIVUM

Vad händer då i framtiden på STACKEN? En billig och "säker" extrapolation vore ju att säga någonting i stil med: –Ja, det blir fler studiebesök, föredrag, samköp, projekt, bla bla bla. Men det vore att fegt smita undan. Det finns en regel, om man så vill, för nyheter: "Impossible! –Fantastic! –So what?" Ta t.ex. räknedosan... för 15 år sedan hade man nog fått svårt att finna finansärer till något så vansinnigt osannolikt som en apparat stor som en almanacka, och som kan beräkna logaritmer med 8 siffrors precision. –Impossible! När de sedan kom blev de flesta rejält imponerade, och som vanligt, en liten skara skrämde. De första gick ut och köpte sig en, och de andra ville förbjuda den, i skolor bl.a. –Fantastic! Idag är det väl få som ägnar någon djupare tanke åt vilka under som försiggår innanför skalet på räknedosan då man står i snabbköpen och jämför literpriset på läskedrycker. –So What?

Magin är synnerligen levande och vital år 1982! Den kände SF-författaren Robert Heinlein formulerade situationen så här år 1950: *A common sense prediction is sure to err on the side of timidity.* Så låt oss släppa alla hämningar och fantisera vilt!

Om tio år kommer STACKEN att:

- ha ett tusen medlemmar. Lågt räknat.
- vara en riksorganisation i stil med Motormännens riksförbund och KAK.
- ha ett fullständigt datoriserat bibliotek: hundratals böcker och tiotusentals artiklar, allt tillgängligt från en terminal och STACKENS password. Stackpointer ges inte längre ut på papper. BYTE och dom andra kommer på teledning en gång/vecka, och artiklarna sorteras automatiskt in i vårt bibliotek.
- vara en remissinstans i alla större utredningar om datorer



- ha en egen kanal, markstation och aktier i den europeiska privatdatasatelliten som sänts upp med den europeiska geoskyttn.
- ha medlemsförmåner i stil med företrädare till de första exemplaren av nya integrerade kretsar (om dom nu ens KALLAS så, då!)

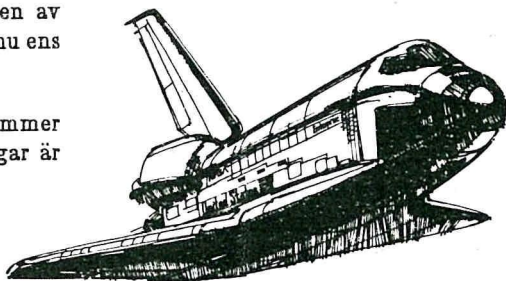
Som sagt, en del av detta kommer att vara fel, men försiktiga gissningar är *garanterat* felaktiga.

VAD KAN STACKEN GÖRA FÖR MIG?

Svar: Inte ett dugg! Nil! ERROR! STACKEN är inte en organisation där medlemmarna blir matade med godbitar, utan att behöva röra en fena. Tvärtom. STACKEN är en sammanslutning av likasinnade, som vill utväxla erfarenheter, kunskaper, samarbeta, lära sig något nytt (av sina kamrater) och skapa. Det enda vi i styrelsen kan bidra med är lite "overhead", såsom bokföring, samordning, hjälpa nya medlemmar tillrätta, sköta korrespondensen med våra systerorganisationer och redigera samman era bidrag till STACKPOINTER. Föreningen är alltså ingen statisk företeelse, utan är vad du vill att den ska vara, och gör den till.

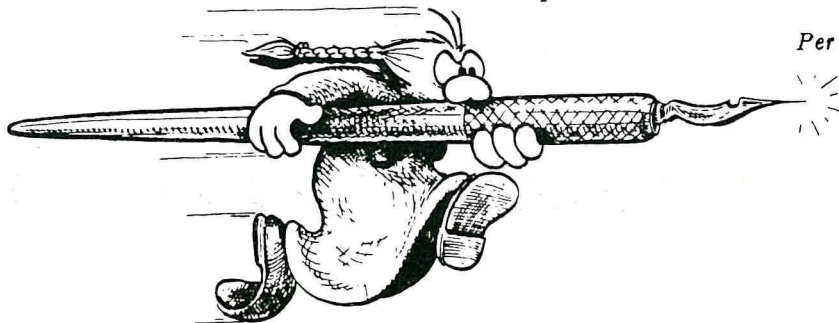
TILL SIST

vill jag å alla föreningens medlemmars vägnar rikta ett varmt tack till alla organisationer, institutioner och företag som hjälpt oss med allt möjligt, och



som ställt upp på studiebesök, gett oss tillgång till överbliven datorkraft, och så vidare. Framför allt då Numerisk Analys och Datalogi (NADA) på KTH, och Stockholms Datamaskincentral (QZ). Utan er hjälp hade inte STACKEN blivit vad den är. Vi hoppas att vi kunnat återgälda förtroendet med lite kunnande, knapptryckande och felfinnande under organiserade former. Och till er som vi besökt på våra studiebesök: Tack för att ni ställt upp och visat era framtidsmaskiner! Vi vet att de snart blir omoderna, så vi kommer tillbaka! Och slutligen vill jag tacka alla medlemmar, utan vilka denna förening inte varit möjlig! Minns di gambles visdomsord: Computare necesse est!

Per Lindberg



Lägesrapport från 16-bitarsgruppen

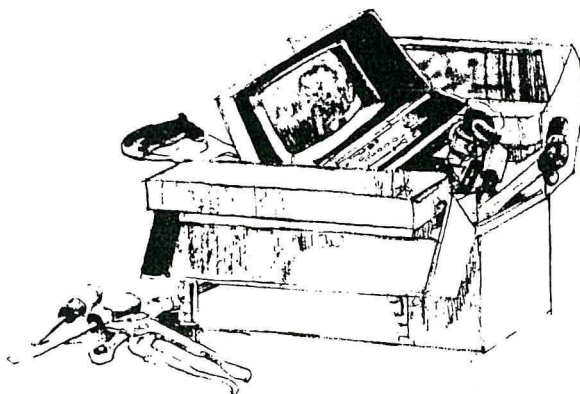
Nu har vi kommit så långt att vi snart ska börja vira en prototyp. Därför kan det kanske vara bra att få veta hur systemet nu ser ut i våra tankar.

Som ni säkert har hört har vi bytt mikroprocessor från Zilog Z-8000 till Motorola 68000. Vi har gjort en del andra ändringar också som ni förmodligen inte har hört talas om.

Följande saker har vi ändrat sedan förra numret av Stackpointer:

- Minneshanteraren (MMU) har flyttats från systembussen till lokalbussen. Detta medför att adressering av lokalbussen kan göras på ett skyddat sätt (via MMU).
- Vi har anpassat lokalbussen till en buss som Motorola har tagit fram (VME-bus). Fördel: Det kommer att finnas färdiga kort att köpa som går att ansluta till vår (och Motorolas) buss.
- CPU-kortet innehåller följande nyheter :
- Anpassningslogik för GPIB (HPIB, IEEE488) CMOS-RAM med batteri för strömförsörjning
- Vi kommer att göra ett anpassningskort för någon 8-bitars vanlig datorbuss för att få tillgång till de många olika sorters I/O-kort som redan finns.

■
Ulf Bilting



LEONARDO DA VINCI INVENTED COMPUTERS!

DAGSLÄGET FÖR AMIS-PROJEKTET

Stackens skärmredigeringsprogram AMIS, verkar nu närma sig slutet av sin utveckling. Arbetet går trögare och trögare allt eftersom det blir färre och färre funktioner som det är angeläget att lägga till. Testversionen av AMIS klarar nu t.ex. av att redigera flera fler samtidigt och att dela skärmen i två halvor som kan användas separat. Förhoppningsvis kan vi inom en inte alltför avlägsen framtid färdigställa den "slutliga" versionen av AMIS.

För närvarande så finns AMIS under operativsystemen TOPS-10 för DEC-10/20, RSTS/E för PDP-11, VMS för VAX och Prime för Prime. AMIS är spridd till ett dussintal maskiner, den mest avlägsna vid Oslo Universitet. Funderingar finns på att implementera AMIS också på NORD-100 och Motorola MC68000-datorer.

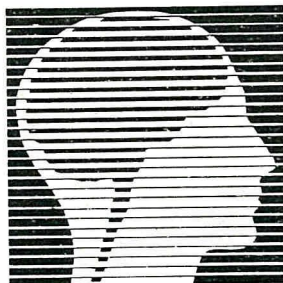
Mera intressant är kanske att Stacken nu har börjat tjäna på sitt arbete med AMIS. I slutet av förra året tecknade Stacken och SIMPROG AB avtal om distribution av AMIS. SIMPROG är ett företag med nära anknytning till NADA och Institutet för Tillämpad Matematik. När SIMPROG inte sluter avtal med Stacken, arbetar de med utveckling av ett SIMULA-system för Prime-datorer.

Avtalet går i korthet ut på att SIMPROG säljer AMIS till kommersiella användare och även tecknar underhållsavtal för de som vill ha sådant. För detta har Stacken fått en engångssumma på 30000 kronor och vi får dessutom 15% på SIMPROGs intäkter av AMIS. Detta öppnar nya möjligheter för Stackens verksamhet: vi kommer att

få råd att investera i egen utrustning, hyra lokal ifall vi skulle vara tvugna att lämna den vi bebor just nu.

Stacken får fortfarande själv sprida AMIS till icke-kommersiella användare så intresserade universitet och ideella organisationer behöver inte betala något speciellt om de skulle vilja ha AMIS.

Lars-Henrik Eriksson



STUDIEBESÖK!

Det har under en längre tid varit tal om ett studiebesök på elektronmusikstudion EMS i Stockholm. Som bekant, har EMS skaffat en VAX, som nyligen installerats. Just nu sysslar man med att lära den spela, och därför har vi avstått från att verkställa vårt studiebesök tills detta är någorlunda klart. Dessutom är det lite extra svårt att få någon från EMS att ställa upp på kvällstid för att guida oss genom Datormusikens Underbara Värld. Men förtvivla icke! Vi har satt ett preliminärt datum för tilldragelsen den 15:e April om aftonen. Mer information kommer på årsmötet, den 1:a April.

Problemhörnan

Nu är det dags för STACKENS medlemmar att ge hjärnan sin välbehövliga motion. Eller, för dom av er som inte använder hjärna: dags att varmköra era AI-funktioner. Snäll som jag är har jag grävt fram tre små problem åt er, hämtade ur KOM-systemet på Nadja:

1

Detta problem finns i Martin Gardners bok *Mathematical Magic Show* och har kanske förekommit i Scientific American.

Två bröder sålde hela sin fårscock och skulle dela lika på pengarna. De fick lika mycket för varje får som antalet får i skocken. De fick betalt i tior och enkronor (< 10 enkronor). Den ena fick en tia mer än den andre.

— Det är orättvist, sade han som fick minst.

— Ok, ta enkronorna också, sade den andre.

— Det räcker inte ändå, sade den andre.

— Ok, då skriver jag ut en check så det blir jämnt, sade han som fått mest.

Hur mycket var checken på?

2

Uthålligheten hos ett (helt) tal definieras som antalet steg som behövs för att reducera det till ett ensiffrigt heltal genom att multiplicera alla dess (decimala) siffror med varandra, och upprepa processen med vad man då fick, tills man har ett ensiffrigt tal.

Låt $m(n)$ beteckna det minsta talet med uthålligheten n . Då är alltså $m(1) = 10$, $m(2) = 25$, $m(3) = 39$ och

$m(4) = 77$. Hur fortsätter serien? Är den ändlig eller oändlig?

3

Professorn i logik hade just tenerat två briljanta unga studenter. När han skulle skriva i tentamensböckerna konstaterade han:

— Jag ser att herrarna är födda samma månad och år men olika dagar. Kan kandidat A då säga vem av er som är äldst?

Det kunde inte A, och B som därpå fick samma fråga kunde inte heller svara. Professorn fortsatte att omväxlande fråga A och B, och till sist sa A:

— Nu vet jag svaret. Jag kan till och med ange vilken dag B är född.

Stackars B visste fortfarande inte svaret men DU bör nu kunna tala om när B är född, år, månad och dag!

Det finns varianter som figurerat i Scientific American dom senaste åren med två matematiker som söker två tal — den ena känner deras summa och den andre deras produkt. Efter några negativa konstateranden kan den ene säga vilka talen är och det räcker för att lösaren i sin tur ska kunna säga det. Den här sortens problem har krånglig logik men inte just någon intressant datastruktur.

Det här var egentligen inga regelrätta programmeringsproblem, men det är alltid trevligt att ha en dator i närheten när man grunnar på matematiska eller logiska problem. Speciellt problem 3 (och liknande) verkar vara i behov av ett program som reder ut logiken åt en.

■

Björn Danielsson

KALLELSE

Till ÅRSMÖTE i datorföreningen STACKEN torsdagen 1 april 1982 klockan 19:00 i sal E7 på Kungliga Tekniska Högskolan.

Förslag till dagordning

- §1. Mötet öppnas
- §2. Val av 2 personer att jämte ordförande justera mötesprotokollet. Justeringsmännen skall tillika tjänstgöra som rösträknare under mötet.
- §3. Val av ordförande för mötet.
- §4. Val av sekreterare för mötet.
- §5. Tillkännagivande av vid mötet uppgjord röstlängd.
- §6. Frågan om mötet är stadgeenligt utlyst.
- §7. Frågan om dagordningens godkännande.
- §8. Framläggande av kassaberättelse och styrelseberättelse.
- §9. Framläggande av revisionsberättelse.
- §10. Frågan om styrelsens ansvarsfrihet under det gångna arbetsåret.
- §11. Val av styrelsemedlemmar:
 - Ordförande
 - Vice Ordförande
 - Sekreterare
 - Kassör
 - Hexmästare
 - Suppleant
 - Redacteur för STACKPOINTER
 - Revisorer med suppleant.
- §12. Val av ledamöter till styrelsevalberedningen.
- §13. Behandling av inkomna motioner:
 - §13.1. (Per Danielsson) Föreningens framtida kontakter med Unga Forskare. (Se motionstext i denna tidning)
 - §13.2. (Hans Nordström) Revidering av stadgar för föreningen. (Se förslagstext i denna tidning)
- §14. Behandling av styrelsens förslag:
 - §14.1. Mötesdagar: 1:a torsdagen i varje månad.
- §15. Behandling av budget för innevarande år.
- §16. Fastställande av medlemsavgifter för 1983.
- §17. Synpunkter på verksamheten för innevarande år och övriga frågor.
- §18. Mötets avslutande.



FÖRSLAG TILL STADGAR FÖR DATORFÖRENINGEN STACKEN

Föreningen

- §1. Föreningen är en partipolitiskt och religiöst obunden, ideell sammanslutning. Styrelsen har sitt säte i Stockholm.
- §2. Föreningens ändamål är att främja intresset för och vidga kunskaper inom datorområdet.
- §3. Föreningens medlemmar skall kallas till möte minst två gånger per år, varav ett skall vara höstmöte och ett skall vara vårmöte.
- §4. Föreningens stadgar kan ändras genom beslut av två efter varandra följande föreningsmöten med minst sextio dagars mellanrum. Ett av dessa skall vara höstmöte eller vårmöte. Nytt förslag till stadgar skall bifogas kallelse till möte. Stadageändring måste bifallas av minst två tredjedelar av de närvarande.
- §5. För föreningens upplösning fordras beslut av två efter varandra följande föreningsmöten med minst sextio dagars mellanrum. Ett av dessa skall vara höstmöte eller vårmöte. Förslag till upplösning av föreningen skall bifogas kallelse till möte. Upplösning av föreningen måste bifallas av minst två tredjedelar av de närvarande. Föreningens återstående tillgångar, efter gäldande av skulder, skall tillfalla ändamål som överenstämmer med föreningens. Beslut härom fattas på det sista mötet.

Medlemmar

- §6. Medlemskap kan beviljas fysisk person som erlagt fastställd medlemsavgift. Till hedersmedlem kan föreningsmöte utse person som förtjänstfullt främjat föreningen eller dess syften.
- §7. Varje medlem har rösträtt med en röst vid föreningens möten. Rösträtt kan ej överlåtas.
- §8. Medlem, som skadar föreningens anseende eller motarbetar dess syfte, kan uteslutas av styrelsen med omedelbar verkan. Uteslutningen skall bekräftas av nästkommande föreningsmöte.

Föreningsmöte

- §9. Föreningsmötet är föreningens högsta beslutande instans.
- §10. Rätt att utlysa föreningsmöte tillkommer styrelsemedlem, revisor eller minst tio medlemmar.
- §11. Kallelse till föreningens möten distribueras till samtliga medlemmar minst fjorton dagar i förväg.
- §12. För att föreningsmöte skall vara beslutsmässigt krävs att minst tjugo medlemmar eller hälften av medlemmarna är närvarande.

- §13. Vid föreningsmöte tages beslut med enkel röstövertikt. Mötesordförande har utslagsröst vid öppen omröstning. Vid lika resultat i slutna omröstning sker lottning.
- §14. Höstmötet skall hållas inom två månader före verksamhetsårets början. Tidpunkt för höstmötet fastställs av föreningsmöte. Under mötet skall alltid följande ärenden behandlas:
- val av styrelse
 - fastställande av firmatecknare
 - val av revisorer
 - fastställande av avgifter
 - fastställande av budget.
- §15. Vårmötet skall hållas inom två månader efter verksamhetsårets slut. Tidpunkt för vårmöte fastställs av föreningsmöte. Under mötet skall alltid följande ärenden behandlas:
- verksamhetsberättelse
 - revisionsberättelse
 - balansräkning
 - ansvarsfrihet för styrelsen.

Styrelsen

- §16. Styrelsen skall bestå av minst tre ledamöter och minst två ersättare. Styrelsen sköter föreningens löpande förvaltning och är ansvarig inför vårmötet.
- §17. Ersättare för avgående styrelseledamot utses, vid behov, av föreningsmöte.
- §18. Rätt att utlysa styrelsemöte tillkommer styrelseledamot eller ersättare.
- §19. Styrelsen är beslutsmässig när samtliga styrelsemedlemmar kallats till styrelsemöte och minst hälften är närvarande.

Redovisning

- §20. Föreningens verksamhetsår sammanfaller med kalenderår.
- §21. Föreningens räkenskaper och styrelsens förvaltning skall granskas av två revisorer.
- §22. Föreningen tecknas av två, av föreningsmöte utsedda styrelseledamöter, var för sig.

Motion

Angående Stackens medlemskap i Unga Forskare.

Eftersom Stacken inte har någon nytta av medlemskapet i Unga Forskare, så yrkar jag på att Stacken utträder ur Unga Forskare. Ytterligare en anledning är att jag ogillar kollektivt medlemskap i en förening.

Grafiska grundrutiner

*If computers are the wave of the future,
then graphic displays are the surfboards.*

*On our glorious Dig-It-All Screens
we mingle the magic of air and fire.*

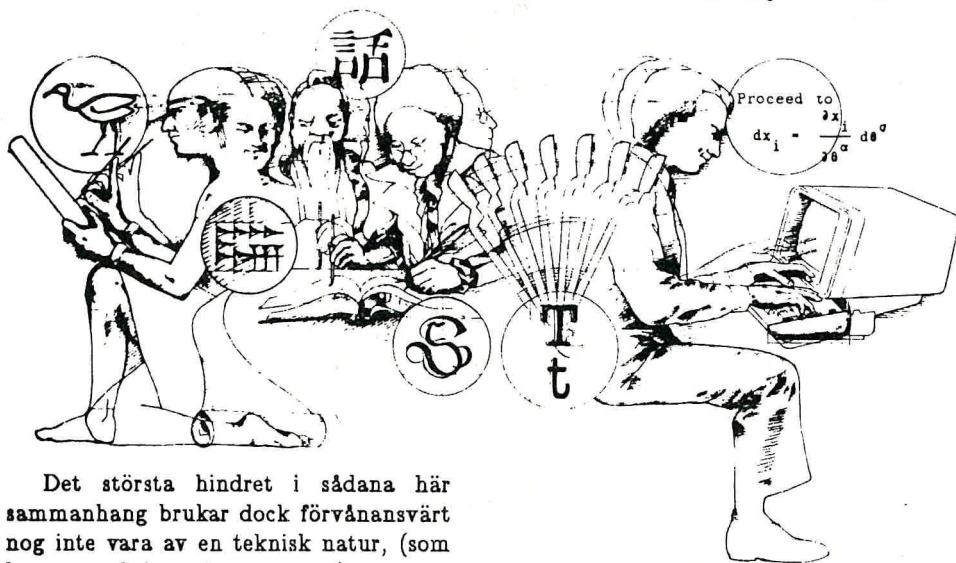
(Ted Nelson)

Alla har vi väl någon gång blivit imponerade av en plottad bild av någon komplicerad funktion i perspektiv, med skynda ytor borttagna, eller blivit fascinerade av en elegant rekursiv kurva. Och kanske tänkt "om jag ändå kunde göra sånt..." med något långtansfullt i blicken.

en psykologisk natur, där man fått för sig att detta är **svårt**! I den här artikeln ska jag försöka visa att det är **lätt**. (Och tycker du fortfarande att det är svårt efter att ha läst den, är det inte ditt fel, utan artikelförfattarens.)

En linje från A till B

Att plotta en bild på en plotter eller en grafisk skärm är oftast inte svårare än att dra ett rakt streck från en punkt till en annan. Alla bilder kan ses som en massa räta linjer. Titta t.ex. på hur en plotter ritar en cirkel. Just det. En massa små räta linjer. Eller den kanske skriver bokstäver? Bara räta linjer här också!



Det största hindret i sådana här sammanhang brukar dock förvånansvärt nog inte vara av en teknisk natur, (som hur man skriver sina program), utan av

Ett grafiskt rutinpaket

Vad man behöver när man ska göra datorgrafik är ett *grafikpaket*, d.v.s. en samling rutiner med vars hjälp man kan dra en linje, rita en cirkel, plotta text och några saker till, som vi ska titta närmare på senare. För programmeraren finns redan ett stort antal färdiga paket att välja på. Många av dessa är väldigt duktiga, och är i stånd till en massa fantastiska trix. Andra, såsom GPGS, har en standardiserad repertoar av rutiner, för att underlätta för den som vill flytta sina program. Flera av dessa, GPGS inräknat, är tyvärr skrivna i FORTRAN, vilket är ett hinder för den som vill skriva sitt grafiska program i något användbart högnivåspråk.

Här ska vi se på ett enkelt grafikpaket som jag skrev i Simula, för att kunna leka med några ideer jag hade. Jag kallade paketet för *ELFIN* (engelska, uttalas "ell-finn"), och sneglade på GPGS för att få en vettig sammansättning av procedurer. Detta hindrade dock inte att jag försökte vara bättre än GPGS på en del punkter. Men mer om detta senare.

Grafiska grundrutiner

Till att börja med behöver vi kunna initialisera skärmen eller plottern, den s.k. "hårdvaran", så att den är redo att ta emot grafisk information. Vi ska med andra ord ställa den i *grafisk mod.* En motsvarande procedur som tar oss ut ur grafisk mod behövs också. Låt oss kalla dessa för *GRAPHICS_ON* och *GRAPHICS_OFF*.

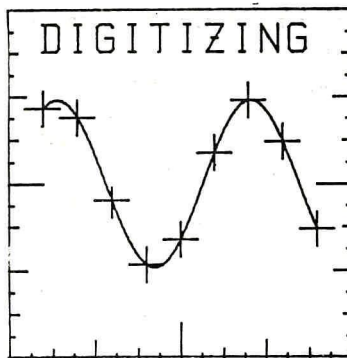
Sedan måste man kunna flytta plotterns penna (på en skärm får vi tänka oss en liknande penna). Vi måste också

kunna bestämma om vi vill ha pennan nedsänkt eller upplyft, d.v.s. om vi ska rita ett streck eller inte. Till detta använder vi procedurerna *DRAWTO(X,Y)* och *MOVE TO(X,Y)*.

Och med dessa fyra procedurer klarar vi oss! Det behövs ingenting mer för att börja plotta kurvor. Allt annat kan vi placera i vårt eget program!

Grafisk inmatning

Hittills har vi bara funderat på grafisk utmatning. Det är dock ofta nödvändigt att även kunna mata in grafiska data från en skärm med inbyggt hårkors, ljuspenna, *mouse*, eller kanske en plotter med möjligheter till digitalisering, eller en riktig *digitizer*, ett digitaliseringsbord avsett enkom för detta.



Vill man ha denna möjlighet, gör man alltså en inläsningsprocedur, låt oss kalla den för *DIGITIZE(X,Y,C)*. Jag har valt att samtidigt läsa in ett tecken *C* från hårdvaran. Detta kan vara användbart som kommando i ett grafiskt program. Dessutom kräver vissa hårdvaror, t.ex. Tektronix att man trycker på en tangent för att läsa in ett koordinatpar.

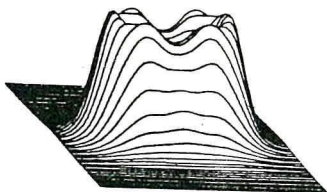
Olika förutsättningar —

samma grafikpaket

Ganska ofta vill man kunna prova ut sitt program på en grafisk skärm, men senare kunna ta ut bilder på en plotter. Detta innebär att ett bra grafikpaket ska kunna rita på *olika* hårdvaror. Alla procedurer i paketet ska alltså göra samma sak, oberoende av vilken utmatningsenhet man använder för tillfället. Här stöter vi på ett problem att lösa: nästan alla fabrikat har olika styrkoder för sina apparater. Det är t.o.m. så illa, att det varierar mellan olika modeller av samma fabrikat. Detta innebär att de grundläggande procedurerna som verkligen pratar med hårdvaran måste ha en separat del för varje hårdvarutyp.

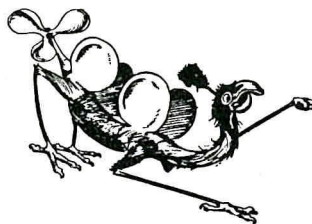
Vi behöver alltså något sätt att tala om för vårt grafiska paket vilken hårdvara som ska användas för tillfället. Detta kan lösas på så sätt att man har olika *bottenpaket* som laddas med vid olika tillfällen. Så fungerar t.ex. GPGS. Detta får tyvärr till följd att man inte kan välja hårdvara, när programmet väl är laddat. I ELFIN har jag i stället valt att ha alla delarna med, och sedan skicka med en kod för vilken hårdvara man vill använda när programmet körs. Detta är en avvägning man måste göra beroende på huruvida man har ont om minne, eller ej. I Simula ser då huvudprogrammet ut så här:

```
EXTERNAL CLASS elfin;
elfin ("TTY:", "HP7221") BEGIN
  < resten av programmet här >
END
```



Den första parametern till ELFIN, "TTY:" anger att jag vill ha utmatningen direkt till terminalen. Annars hade jag kunnat ange ett filnamn här, så hade jag i stället fått all utmatning sparad som en fil. Den andra parametern, "HP7221", anger att jag vill ha styrtecken som passar en Hewlett-Packard 7221 Plotter.

Naturligtvis är det smidigt att kunna ange utfil och hårdvarutyp först när man kör programmet. Därför kan man antingen låta huvudprogrammet fråga användaren om dessa, och sedan skicka med dessa som parametrar till ELFIN, eller också inte skicka med någonting alls. Då sköter ELFIN själv om den saken, och frågar användaren efter utfil och hårdvara. Så här: `elfin("","")` BEGIN ... END;



An ork of Oz

I vissa fall finns det ett problem till: terminalen eller plottern kan ha ett valbart *command character*, som ska inleda en grafisk sekvens. Detta kan ibland vara praktiskt, då man vill koppla ihop flera olika enheter, eller då man har program som använder command character till någonting annat. Vilket tecken som ska användas brukar i så fall vara inställbart genom att man pillar på en omkopplare på baksidan av apparaten. I ELFIN har jag proceduren SET_CMD_CHAR(C) som talar om vilket styrtecken som ska inleda en grafisk sekvens för dessa hårdvaror.

Fler grundrutiner för ökad bekvämlighet

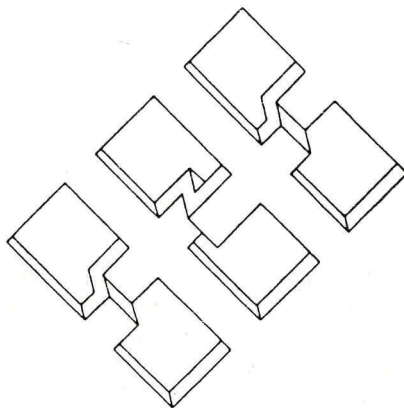
Eftersom jag är lat, vill jag ha en del saker i mitt grafikpaket som jag annars måste ha i mina program som ska använda det. Sådana saker är t.ex. att kunna rita cirkelbågar och plotta text. För cirkelbågar behöver man en procedur `ARC(XCENTER,YCENTER,ANGLE)`, som ritar en cirkelbåge från pennans position med `XCENTER`, `YCENTER` som mittpunkt och `ANGLE` grader lång. Tecknet på `ANGLE` kommer alltså att bestämma åt vilket håll bågen ska ritas. I många fall vill man ange vinkeln i radianer istället. För detta finns hjälprutinen `ARCRAD` med samma parametrar. Ytterligare en användbar hjälprutin är `CIRCLE(R)`, som ritar en hel cirkel med radien `R` runt pennans aktuella position.

De allra flesta hårdvaror har en inbyggd funktion för att plotta text. Om man ska använda olika hårdvaror är detta dock oanvändbart, eftersom de alla har olika format, och på en del hårdvaror saknas möjlighet att få olika storlek på texten, o.s.v. Därför behövs två procedurer: `DRAWTEXT(T)`, som ritar texten `T`, och `TEXTSTYLE(X,Y,SLANT,DIRECTION)` som sätter teckenstorleken till `X`, `Y`, kursiveringsvinkel till `SLANT` och riktning på texten till `DIRECTION`. Med dessa klarar man sig mycket långt. Ytterligare finesser vore att kunna välja mellan olika fonter, och att även kunna rita sina egna fonter.

```
!"#$%&'()*+,-./01234567
89:;<=>?@ABCDEFGHIJKLMNO
PQRSTUVWXYZÄÖÅ^_`abcdefg
hijklmnopqrstuvwxyzäöå~
```

Variable character SIZE, SLANT and

DIRECTION

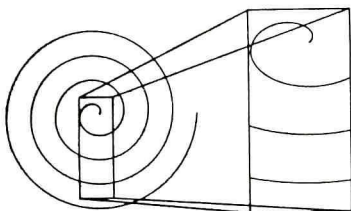


Skalningsrutiner ger ökad flexibilitet

Nu har vi ett paket som är nästan helt oberoende av vilken hårdvara som för tillfället används som utmatningsenhet. Det enda som återstår för att vi ska bli totalt oberoende, är att ha samma gradering av koordinatsystemet oavsett vilka koordinater hårdvaran använder. Detta skulle kunna göras så, att grafikpaketet räknar om alla koordinater beroende på hårdvarukod. Man kan t.ex. göra så att koordinaterna alltid ligger inom 0 - 100 eller 0 - 1. Men detta är inte heller riktigt bra.

För det första vill användaren kanske kunna komma åt varje punkt för sig, och detta blir olika för varje hårdvara. Lätt fixat: vi inför funktionerna `XMAX` och `YMAX` som lämnar tillbaka den aktuella hårdvarans max-koordinater.

För det andra vill man som användare själv kunna bestämma vilka koordinater man vill röra sig inom. För detta används proceduren `SCALE(X0,X1,Y0,Y1)`. Med hjälp av denna kan grafikpaketet själv räkna om alla koordinater så att gränserna är `X0 - X1` och `Y0 - Y1`. Ett exempel: Antag att vi vill ha våra koordinater mellan -1 och +1 i både x-led och y-led. Lätt fixat: vi anropar `SCALE(-1,1,-1,1)`; och det fungerar. 17



Till sist vill man gärna kunna placera sin "bild" var som helst på skärmen eller papperet. För att göra detta, skapar man sig en s.k. *viewport*. Med proceduren `VIEWPORT(X0,X1,Y0,Y1)` får man all utmatning (fortfarande angivna i de koordinater som angavs med `SCALE`) att hamna mellan $X0 - X1$ och $Y0 - Y1$.

Nu har vi plötsligt två procedurer som gör linjär skalning av vad vi plottar. Vad är då skillnaden? Jo, enkelt uttryckt, med `SCALE` bestämmer vi vilka koordinater vi vill röra oss inom (world coordinates), medan `VIEWPORT` säger var på papperet/skärmen vår "värld" ska hamna (screen coordinates).

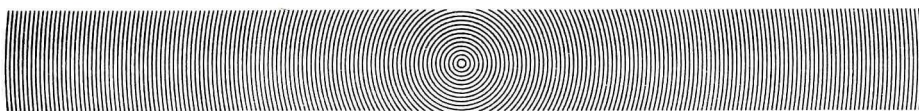
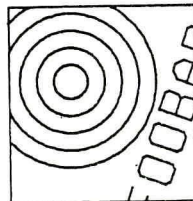
Andra saker som kan vara bra att ha

Ytterligare ett par procedurer ska nämnas innan denna sammanställning är komplett. I en del fall är det bekvämt att kunna ange relativa koordinater i stället för absoluta. Därför är procedurerna `MOVER(XR,YR)` och `DRAWR(XR,YR)` bra att ha. En del plottar kan själva mata fram nytt papper, och alla skärmar kan radera skärmen. Detta utnyttjas i proceduren `NEWPAGE`.

Fler finesser som man vill kunna utnyttja är färg, att kunna sudda linjer, och att kunna välja penna i en plotter med flera pennor. Alla dessa går att samla i proceduren `NEWPEN(P)`, som gör lite olika saker, beroende på hårdvara. På en plotter som har olika pennor, väljs penna nummer P. Har man en färgskärm, väljs färg nummer P. Om P är 0, sätts pennan tillbaka på en plotter som kan det, och en skärm som kan radera sätts i "suddningsmod". Om P är 1, plottar alla typer av hårdvara vanliga streck. På en färgskärm blir alltså 0=svart och 1=vitt. (Eller grönt, beroende på färgen på fosfor i bildskärmen!)

En finess som brukar betraktas som lite mer avancerad, är klippning av linjer som kommer utanför en bestämd ram. Detta är mycket trevligt att ha, ifall man inte i förväg vet om strecken kommer att hamna utanför skärmen. Äldre typer av grafisk utrustning, t.ex. Tektronix, får fnatt om man försöker rita utanför kanten, och hämnas genom att rita en massa skräplinjer kors och tvärs över hela rutan. Proceduren `CLIP(X0,X1,Y0,Y1)` skär av alla linjedelar som går utanför ramen $X0 - X1, Y0 - Y1$.

CLIPPING



Handskakning

Om man, som brukligt är i vissa kretsar, kör helt asynkront kan det bli misär om inte plottern hinner med att rita allt den får ta emot av datorn. De flesta plottrar har visserligen en inbyggd buffer som köar upp inkommande tecken, men när den är fylld, så går det ändå åt skogen. Därför måste man antingen köra med en mycket låg överföringshastighet (och inte ens då är det riktigt säkert att plottern hinner med i alla lägen!), eller också ha någon sorts protokoll för överföringen, där man frågar plottern hur den mår för vart 100:e tecken eller så. Men sker utmatningen på en fil, är det rätt meningslöst att fråga filen samma sak...

Sammanfattning

Med det paket av grafiska grundrutiner jag beskrivit, kan man enkelt rita bilder med ett program. Paketet tar självt hand om sådana saker som man inte ska behöva bekymra sig om varje gång man vill göra något med grafik. Sådana automagiska saker innefattar bl.a. att välja typ av hårdvara, att fråga användaren om vart utmatningen skall skickas (fil eller terminal-ledning), och även att sköta handskakning där så behövs.

Här har bara beskrivits grafik som ligger i ett plan. Vill man göra mer avancerad grafik i 3-D, måste man ha en extra koordinat i alla rutiner, och sedan projicera alla linjer på ett plan, innan man plottar. Är man riktigt ambitiös, kan man sedan bygga på med rutiner för rotation av föremål och

borttagning av skymda ytor m.m. Men i botten på alla paket, vare sig enkla eller avancerade, måste finnas ett par små simpla procedurer, såsom MOVETO och DRAWTO, för att kunna dra ett streck från A till B.

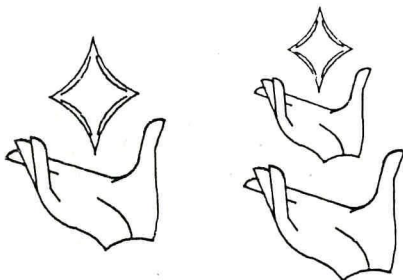
Och därmed är vi tillbaka där vi började; börja skriv ett par basrutiner, och hela datorgrafikens underbara värld är din!

För den som vill lära sig mer om datorgrafik i allmänhet, kan jag varmt rekommendera boken *Principles of Interactive Computer Graphics* av Newman & Sproull (McGraw-Hill). Den är ett "standardverk" inom ämnet, och är välstrukturerad och lättläst. Boken innehåller bl.a. beskrivning av de grundläggande ideerna, vilkas motsvarighet återfinns i några av de rutiner som jag nyss beskrivit. Den innehåller också beskrivningar av alla grundläggande algoritmer, vilket gör den till rena uppslagsverket för den som vill syssla med datorgrafik. Boken har även en massa exempel, skrivna i PASCAL, så att de är väldigt lätta att läsa och förstå.

Per Lindberg

Referenser:

Newman & Sproull: *Principles of Interactive Computer Graphics*
McGraw-Hill



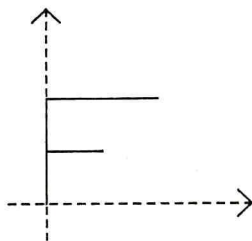
Transformerings och plottning av text

En av de saker man gärna vill ha med på plottbilder är text, som förklarar vad bilderna föreställer. Det finns plottrar, som kan rita text om man skickar särskilda styrkoder till dem, men om man av någon anledning inte vill utnyttja den inbyggda texten eller har en plotter, som inte själv kan rita text, så måste man skriva särskilda rutiner för detta i sitt program eller plottpaket.

Bokstävernas utseende

All text består av bokstäver och andra tecken, som i sin tur består av streck och kurvor. Eftersom de flesta plottrar bara kan rita streck så måste även kurvorna ritas med streck. Ju fler streck desto jämnare blir kurvorna och desto längre tid tar det att rita dem.

Ett stort F kan ritas med två linjer. En som börjar i (0,0), går till (0,2) och fortsätter till (2,2) och en som går från (0,1) till (1,1).



Vanligt F

Att lära datorn hur bokstäverna ser ut är ett tidsödande och pilligt arbete. Oftast ritas man först upp bokstäverna på ett papper och matar sedan in i datorn var strecken börjar och slutar. Till detta

20 behöver man ett specialskrivet program,

som läser in koordinaterna för streckens ändpunkter, ritas upp bokstäverna, så man ser att de blev rätt inmatade, och sedan sparar koordinaterna på en fil.

Om man lagrar koordinaterna i binärt format på en skivminnesfil, som text, som en LISP-lista, som DATA-satser i ett BASIC-program eller på annat sätt beror framför allt på vilket programspråk och vilken dator man arbetar med.

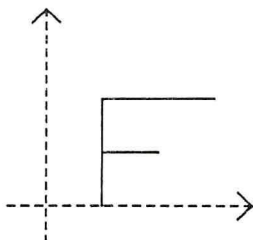
Grundläggande transformationer

När man har definierat en font (en uppsättning bokstäver och andra tecken) vill man kanske rita likadana bokstäver i annan storlek, lutande bokstäver eller rita hela texten på en annan ledd. I stället för att knappa in nya fonter för varje tillfälle kan man låta den existerande fonten genomgå ett antal enkla transformationer. De operationer man behöver är translation, skalning, skjuvning och rotation. Eftersom samtliga dessa avbildningar är affina (en rät linje avbildas på en rät linje) så behöver man inte transformera strecken i bokstäverna utan bara deras ändpunkter.

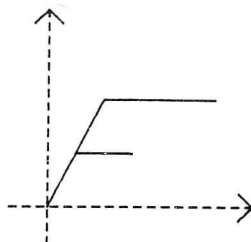
Translation av text

Den första transformationen bokstäverna utsätts för är parallellförskjutning i x-axelns positiva riktning, så att inte alla bokstäverna i texten ska ritas över varandra på samma ställe. Förskjutningen i förhållande till textens början bestäms av $x_{\text{parallell}}$, som ökas på med en dryg bokstavsbredd efter varje utskriven bokstav. Transformationen kan skrivas

$$A : (x, y) \mapsto (x + x_{\text{parallell}}, y)$$



Parallellförskjutet F

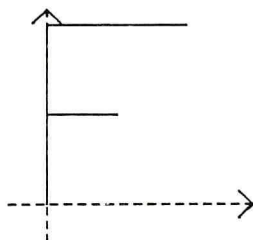


Framåtlutande F

Skalning av text

Förstorade eller förminskade bokstäver får man genom att multiplicera alla avstånd i x -led med x_{skal} och i y -led med y_{skal} . Om man inte vill ha någon storleksändring sätter man båda konstanterna till ett. Transformationen ges av

$$B : (x, y) \mapsto (x x_{skal}, y y_{skal})$$

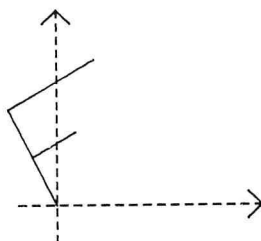


Förstorat F

Rotation av text

Om man vill plotta text på annan ledd än vågrätt måste hela texten vridas runt den punkt där den börjar. Att vrida texten en vinkel ϕ_{vrid} är detsamma som att multiplicera varje punkt med ett komplext tal med absolutbeloppet ett och argumentet ϕ_{vrid} . Transformationen blir

$$D : (x, y) \mapsto (x \cos \phi_{vrid} - y \sin \phi_{vrid}, \\ y \cos \phi_{vrid} + x \sin \phi_{vrid})$$



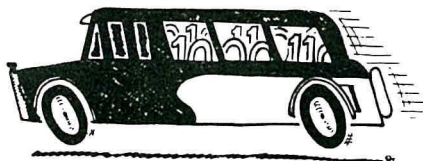
Bakåtvridet F

Skjuvning av text

För att få lutande bokstäver måste man vrida de lodräta staplarna åt höger eller vänster utan att samtidigt vrida vågräta linjer. Om skjuvningsvinkeln ϕ_{skjuv} mellan y -axeln och staplarna räknas positiv åt höger ges transformationen av

$$C : (x, y) \mapsto (x + y \tan \phi_{skjuv}, y)$$

Jan Michael Rynning



Rekursiva kurvor

eller

Hur man gör datorkonst utan
att egentligen anstränga sig

*Here's to pure mathematics
- may it never be of any use to anybody!*

(Old famous toast)

*Hofstadter's Law: It always takes longer than you expect,
even when you take into account Hofstadter's Law*

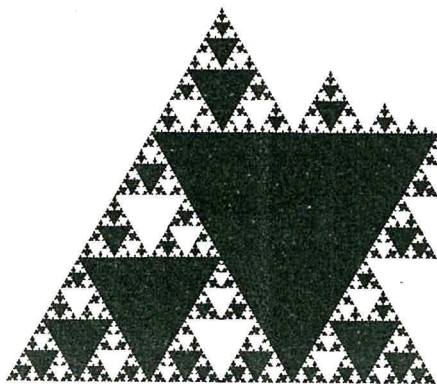
Tänk om det fanns en sluten kurva i planet som var oändligt lång, men ändå inneslöt en ändlig yta! Tänk om det fanns en kurva som var överallt kontinuerlig, men ingenstans deriverbar! Tänk om det fanns kurvor som kunde fylla ut planet fullständigt, så att det inte gick att stoppa in fler linjer där oavsett hur tunna de är! Eller tänk om man hade en figur där varje del såg likadan ut, oavsett hur stor delen är! Tänk om...

Nej, det är inte Science Fiction, trots att det står "Tänk om..." lite varstans. Det är Science Fact, alltihop. Allt detta går att åstadkomma med *rekursiva kurvor*.

Recept på en rekursiv kurva med vaniljsmak

För att göra en rekursiv kurva utgår man från en eller ett par räta linjer. Sedan sätter man upp en regel för hur en rät linje ska delas in i flera räta linjer, t.ex. ett sick-sackmönster, som börjar och slutar i samma punkter som den ursprungliga räta linjen. Denna regel

applicerar man sedan även på *var och en* av de nya mindre räta linjerna. Vill man kunna rita kurvan i verkligheten, måste man även bestämma ett maximalt rekursionsdjup, så att rekursionen inte fortsätter i all oändlighet. När detta djup är nått, *ritar man ut* linjen i stället för att dela in den i fler ännu mindre bitar.



C-kurvan

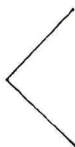
Låt det här för teoretiskt? Låt oss då se på ett enkelt exempel, *C-kurvan*. Detta är en av klassikerna, som finns beskriven i bl.a. HAKMEM (se litteraturföreteckningen).

Man tager sig en rät linje mellan A och B. Sedan tar man fram regeln för C-kurvan som säger att varje linje skall ersättas av två lika långa linjer, i rät vinkel mot varandra, med vinkelspetsen åt vänster. Upprepas rekursivt.

Då ska vi se: en rät linje



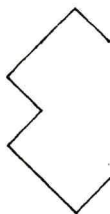
Dela in den i två lika långa linjer i rät vinkel mot varandra, med vinkelspetsen åt vänster



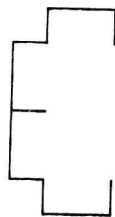
Men för att rita *de* linjerna, måste vi på samma sätt dela in dem i två



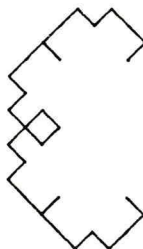
Men för att göra *det*...



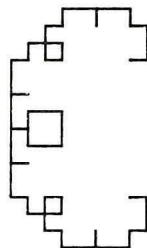
Och så vidare...



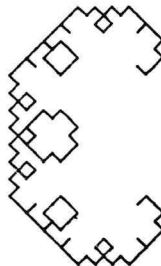
Och så vidare...



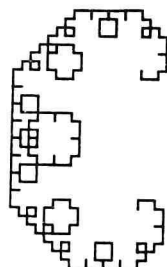
Och så vidare...



Och så vidare...



(Nu börjar man ana varför den heter C-kurvan...)



Om vi ska fortsätta så här, kommer kurvan aldrig att bli klar. Oändlig rekursion är visserligen intressant ur rent matematisk synvinkel, men vill man ha ett resultat någon gång, måste man även införa ett *avbrottsvillkor*. I detta sammanhang är det praktiskt att avbryta när rekursionen nått ett visst förutbestämt djup. I exemplet vi just såg på, avbröts rekursionen efter 0,1,2,3,4,5,6,7 och 8 rekursiva anrop.

Konsten att fylla en yta

En variant av C-kurvan, *The Dragon-curve*, får man då man omväxlande låter den rätta vinkeln sticka ut åt vänster och höger. Hur den ser ut, kan beskådas i förra numret av STACKPOINTER, tillsammans med det LISP-program som ritade den.

"Drak-kurvan" är, till skillnad från sin kusin C-kurvan, en *ytuppfyllande*

kurva. Denna egenskap uppstår givetvis bara vid oändlig rekursion. Icke desto mindre blev den första upptäckten av en sådan kurva i slutet av artonhundratalet av matematikern Giuseppe Peano rätt uppmärksammas i matematiska kretsar. Peanos kurva är en linje som fyller en kvadrat mellan (0-1, 0-1). För att åstadkomma detta, krävs att man har en kurva som 1. slingrar sig till varje punkt på ytan, 2. när rekursionen går mot oändlighet, måste varje punkt på kurvan gå mot en gränsvärdespunkt. Peano utgick från en enkel kurva som ersätts av nio mindre kurvor, hopsatta med åtta extra små linjer. De nio småkurvorna är alla kopior av den ursprungliga, fast de kan vara spegelbilder av denna.

Peano-kurva i CBASIC

Om vi ska försöka rita en peano-kurva i CBASIC (TDL:s "12 k-BASIC"), kan vi utnyttja användardefinierade funktioner

NOW!
THE BATTLE OF GIANTS!
★The Champ★ ! Challenger!



The Hallerith
MONSTER
ALIAS

FORTTRAN

VS.



The Tree
WIZARD
ALIAS

LISP

för att kunna göra rekursiva anrop. Dessa har nämligen finessen att parametrarna vid varje anrop blir lokala variabler i varje nytt anrop, även om det är rekursivt. Här behöver vi 2 parametrar: I% för aktuellt rekursionsdjup och C% för att ange vilken av de 4 spegelbilderna som önskas. Dessutom behöver vi en lokal variabel, P%, som ska hålla rätt på hur långt vi kommit i ritandet av kurvan på denna

rekursionsnivå. Eftersom parametrar blir lokala variabler, kan vi skapa en sådan genom att ha en extra dummy-parameter. Kanske inte så elegant, men det är ju BASIC, och det fungerar. Hur de 4 olika varianterna av grundkurvan ska se ut, definieras i A\$(1..4). Genom att bara ändra dessa, kan vi enkelt rita andra liknande rekursiva kurvor med samma program.

```

100 'Definition of the mutually recursive parts of a Peano curve
110 A$(1%)="1↑2↑1>3v4v3>1↑2↑1"
120 A$(2%)="2↑1↑2<4v3v4<2↑1↑2"
130 A$(3%)="3v4v3>1↑2↑1>3v4v3"
140 A$(4%)="4v3v4<2↑1↑2<4v3v4"
150 '+-----+
160 '!                               LINE-DRAWING PROCEDURE                               !
170 '+-----+
180 DEF FNDW%(X%,Y%)
190 IF X%<>0% THEN FOR Z%=2%*XO% TO 2%*(XO%+X%) STEP SGN(X%)
   : SETDOT YO%,Z% : NEXT Z% : XO%=XO%+X% : FNRETURN
200 IF Y%<>0% THEN FOR Z%=YO% TO YO%+Y% STEP SGN(Y%)
   : SETDOT Z%,2%*XO% : NEXT Z% : YO%=YO%+Y% : FNRETURN
210 FNEEND
220 '+-----+
230 '!                               RECURSION HANDLER                               !
240 '+-----+
250 DEF FNQ%(I%,C%,P%)
260 IF I%=0% THEN FNRETURN
270 FOR P%=1% TO 17%
280 C$=MID$(A$(C%),P%,1%)
290 IF C$="<" THEN D%=FNDW%(-S%,0%) ELSE
   IF C$=">" THEN D%=FNDW%(S%,0%) ELSE
   IF C$="v" THEN D%=FNDW%(0%,S%) ELSE
   IF C$="↑" THEN D%=FNDW%(0%,-S%) ELSE
   D%=FNQ%(I%-1%,INSTR("1234",C$),0%)
300 NEXT P%
310 FNEEND
320 '+-----+
330 '!                               I N I T I A L I Z E                               !
340 '+-----+
350 SO%=54% ' Max sidelength
360 ? CHR$(12%);

```

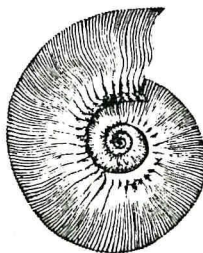
```

370 '+-----+
380 '!               M A I N   L U P E               !
390 '+-----+
400 ? CUR(0%,0%);CHR$(22%);"Rekursionsgrad"; : INPUT I%
410 ? CHR$(12%);CUR(0%,0%);"Peano-kurva av ordning";I%;
420 FOR X%=0% TO 159% : SETDOT 3%,X% : SETDOT 71%,X% : NEXT X%
430 FOR Y%=4% TO 70% : SETDOT Y%,0% : SETDOT Y%,159% : NEXT Y%
440 XO%=12% : YO%=64%
450 S%=80%/(3%**I%) : XO%=XO%+S%/3% : YO%=YO%-S%/3%
460 IF S%<2% THEN 400' Scale too small
470 D%=FNQ%(I%,1%,0%)' Begin with a type-1 curve.
480 GOTO 400
490 '+-----+
500 '!               E N D   M A I N   L U P E               !
510 '+-----+

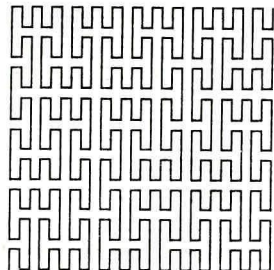
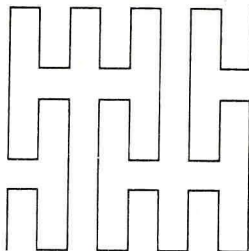
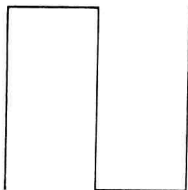
```

I denna implementation av CBASIC, nämligen METRIC-85 CBASIC, finns satsen SETDOT R%,K% som i likhet med ABC80-BASIC sätter en grafisk pixel på rad R%, kolumn K%. Satsen PRINT kan förkortas med ett frågetecken. PRINT CUR(R%,K%); sätter markören på rad R%, kolumn K% och PRINT CHR\$(22%); renser resten av den rad som markören står på. Rester av programmet bör vara skåpmat för den som har sett någon extended BASIC. (I nästa version av CBASIC kan man strunta i dummy-parametern vid anropet av FNQ%)

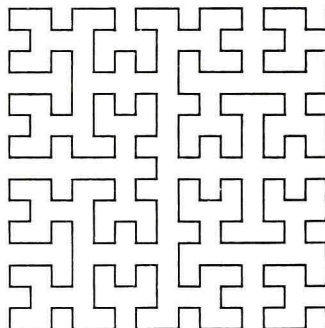
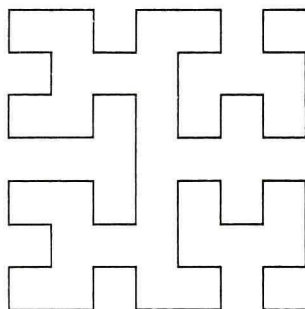
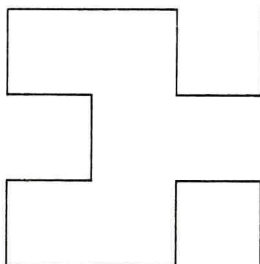
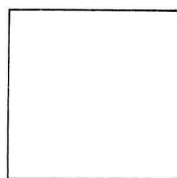
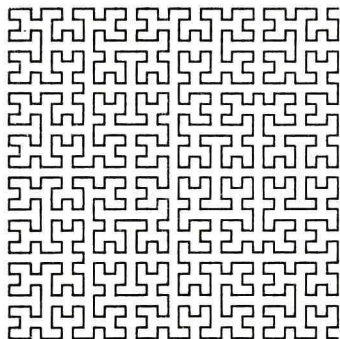
Så här ser de första tre rekursionsnivåerna ut om man plottar dem med en pennplotter:



Naturlig
rekursion?



Hilbert-kurvan



En av de klassiska ytuppfyllande kurvorna är *Hilbert-kurvan*, uppkallad efter matematikern David Hilbert. Den påminner lite om Peano-kurvan. Här utgår man från tre lika långa linjer i rät vinkel mot varandra, så att de bildar ett C. Sedan ersätter man denna C-formade figur med fyra likadana figurer, hälften så stora, och sammanbundna med tre extra linjer. De fyra nya C-formade figurerna ska dessutom vara vridna på ett visst sätt, så att den nya kurvan "slingrar" sig runt den tidigare. Genom att hacka lite i programmet ovan, kan vi få det att rita Hilbert-kurvor istället:

```
110 A$(1%)="4<1v1>2"
120 A$(2%)="3†2>2v1"
130 A$(3%)="2>3†3<4"
140 A$(4%)="1v4<4†3"
```

```
270 FOR P%=1% TO 7%
```

Dessutom bör man ändra startpunkt och längden av en linje.

Så här ser de första fyra rekursionsnivåerna av hilbert-kurvan ut:

Nu är som tidigare sagts inte BASIC något lämpligt språk för att rita rekursiva kurvor, eftersom rekursion inte finns i vanilj-BASIC. Ett språk som lämpar sig bättre är PASCAL. Så här kan det då se ut, om vi gör en procedur för varje delkurva, och låter dem anropa varandra rekursivt:

```
PROGRAM Hilbert(input,output);
```

```
CONST n=4; h0=512;
```

```
VAR i,h,x,y,x0,y0: INTEGER;
```

```
PROCEDURE A(i:INTEGER);
```

```
BEGIN
```

```
IF i > 0 THEN BEGIN
```

```
D(i-1); x:=x-h; drawto(x,y);
```

```
A(i-1); y:=y-h; drawto(x,y);
```

```
A(i-1); x:=x+h; drawto(x,y);
```

```
B(i-1);
```

```
END;
```

```
END;
```

```
PROCEDURE B(i:INTEGER);
```

```
BEGIN
```

```
IF i > 0 THEN BEGIN
```

```
C(i-1); y:=y+h; drawto(x,y);
```

```
B(i-1); x:=x+h; drawto(x,y);
```

```
B(i-1); y:=y-h; drawto(x,y);
```

```
A(i-1);
```

```
END;
```

```
END;
```

```
PROCEDURE C(i:INTEGER);
```

```
BEGIN
```

```
IF i > 0 THEN BEGIN
```

```
B(i-1); x:=x+h; drawto(x,y);
```

```
C(i-1); y:=y+h; drawto(x,y);
```

```
C(i-1); x:=x-h; drawto(x,y);
```

```
D(i-1);
```

```
END;
```

```
END;
```

```
PROCEDURE D(i:INTEGER);
```

```
BEGIN
```

```
IF i > 0 THEN BEGIN
```

```
A(i-1); y:=y-h; drawto(x,y);
```

```
D(i-1); x:=x-h; drawto(x,y);
```

```
D(i-1); y:=y+h; drawto(x,y);
```

```
C(i-1);
```

```
END;
```

```
END;
```

```
BEGIN
```

```
GraphicsOn;
```

```
i:=0; h:=h0; x0:=h DIV 2; y0:=x0;
```

```
REPEAT
```

```
i:=i+1; h:=h DIV 2;
```

```
x0:=x0+(h DIV 2); y0:=y0+(h DIV 2);
```

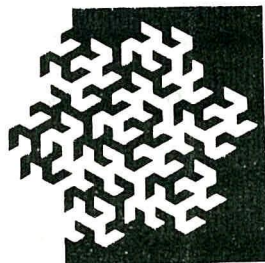
```
x:=x0; y:=y0; moveto(x,y);
```

```
A(i);
```

```
UNTIL i=n;
```

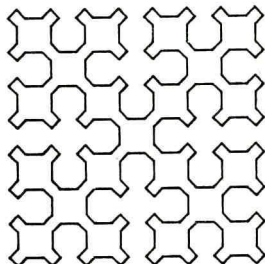
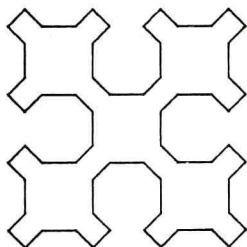
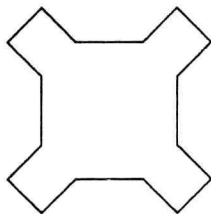
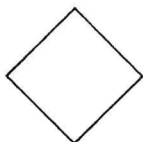
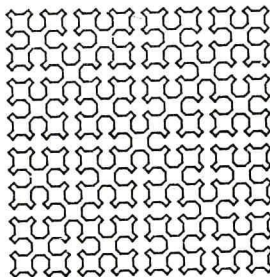
```
GraphicsOff;
```

```
END.
```



Sierpinski-kurvan

En vackrare variant av Hilbert-kurvan är Sierpinski-kurvan, som är lite mer komplicerad. Här utgår man från en kvadrat ställd på ena hörnet, och ersätter varje hörn med en av 4 olika omvägar. Så här ser den ut:



Eller i PASCAL:

```

PROGRAM Sierpinski(input,output);
(* Plotta Sierpinski-kurvor av ordningen 1 till n *)
CONST n=4; h0=512;
VAR i,h,x,y,x0,y0: INTEGER;

PROCEDURE A(i:INTEGER);
BEGIN
  IF i > 0 THEN BEGIN
    A(i-1); x:=x+h; y:=y-h; drawto(x,y);
    B(i-1); x:=x+2*h; drawto(x,y);
    D(i-1); x:=x+h; y:=y+h; drawto(x,y);
    A(i-1);
  END;
END;

PROCEDURE B(i:INTEGER);
BEGIN
  IF i > 0 THEN BEGIN
    B(i-1); x:=x-h; y:=y-h; drawto(x,y);
    C(i-1); y:=y-2*h; drawto(x,y);
    A(i-1); x:=x+h; y:=y-h; drawto(x,y);
    B(i-1);
  END;
END;

PROCEDURE C(i:INTEGER);
BEGIN
  IF i > 0 THEN BEGIN
    C(i-1); x:=x-h; y:=y+h; drawto(x,y);
    D(i-1); x:=x-2*h; drawto(x,y);
    B(i-1); x:=x-h; y:=y-h; drawto(x,y);
    C(i-1);
  END;
END;

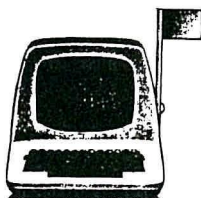
```

```

PROCEDURE D(1: INTEGER);
BEGIN
  IF 1 > 0 THEN BEGIN
    D(1-1); x:=x+h; y:=y+h; drawto(x,y);
    A(1-1); y:=y+2*h; drawto(x,y);
    C(1-1); x:=x-h; y:=y+h; drawto(x,y);
    D(1-1);
  END;
END;

BEGIN
  GraphicsOn;
  i:=0; h:=h0 DIV 4; x0:=2*h; y0:=3*h;
  REPEAT
    i:=i+1; x0:=x0-h; h:=h DIV 2; y0:=y0+h;
    x:=x0; y:=y0; Moveto(x,y);
    A(i); x:=x+h; y:=y-h; Drawto(x,y);
    B(i); x:=x-h; y:=y-h; Drawto(x,y);
    C(i); x:=x-h; y:=y+h; Drawto(x,y);
    D(i); x:=x+h; y:=y+h; Drawto(x,y);
  UNTIL i = n;
  GraphicsOff;
END.

```



Utflekade flingor

Ett annat vackert exempel på en rekursiv kurva är *the Snowflake Curve*, som uppfanns av Helge von Koch. Snöflinge-kurvan fyller visserligen inte upp en yta, men har andra intressanta egenskaper: Den är av oändlig längd men omsluter ändå en ändlig yta(!), den är överallt kontinuerlig, men ingenstans deriverbar.

Vid konstruktionen av en snöflinga utgår vi från tre rätta linjer som bildar en liksidig triangel. Sen byter vi ut varje linje mot fyra, så att det sticker ut en ny liten (liksidig) triangel, en tredjedel så stor, på var och en av triangelns sidor. Och på var och en av dessa sidor...

Eller i LISP med TURTLE-GRAPHICS:


```
(defun draw-snowflake-curve (depth)
```

```
  (drawpart depth 1)
```

```
  (turn-left 120)
```

```
  (drawpart depth 1)
```

```
  (turn-left 120)
```

```
  (drawpart depth 1))
```

```
(defun drawpart (depth length)
```

```
  (cond ((zerop depth) (walk length))
```

```
  (t (drawpart (sub1 depth) (quotient length 3))
```

```
      (turn-right 60)
```

```
      (drawpart (sub1 depth) (quotient length 3))
```

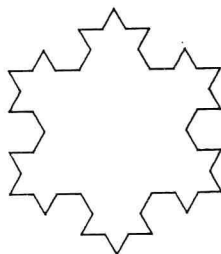
```
      (turn-left 120)
```

```
      (drawpart (sub1 depth) (quotient length 3))
```

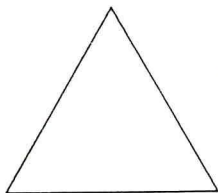
```
      (turn-right 60)
```

```
      (drawpart (sub1 depth) (quotient length 3))))))
```

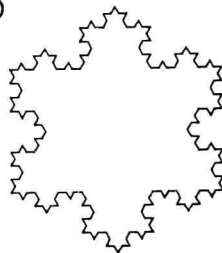
```
(draw-snowflake-curve 2)
```



Så här ser det då ut med anrop av
(draw-snowflake-curve 0)

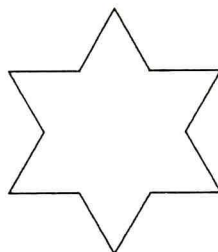


```
(draw-snowflake-curve 3)
```

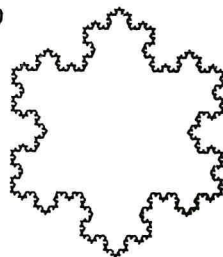


```
(draw-snowflake-curve 1)
```

ger första rekursionsgraden:



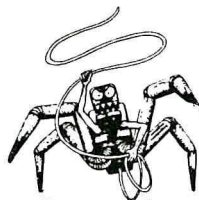
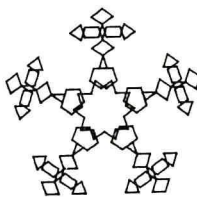
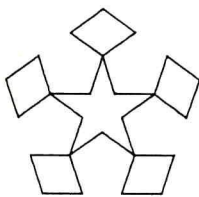
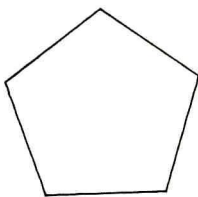
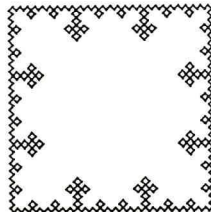
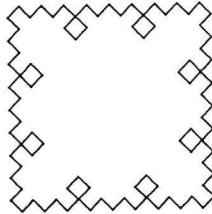
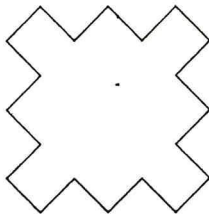
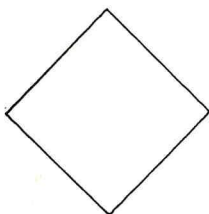
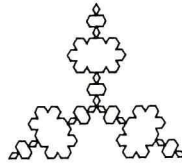
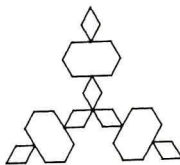
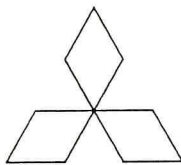
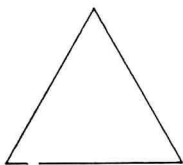
```
(draw-snowflake-curve 4)
```



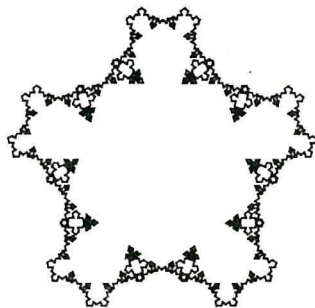
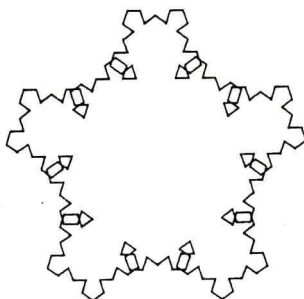
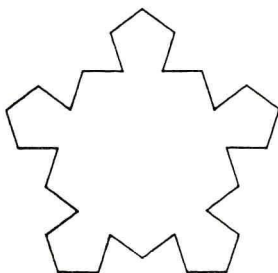
Just *turtle-graphics* är en mycket lämplig miljö att plotta rekursiva kurvor i. Och LISP är en mycket lämplig miljö för grafik över huvudtaget. I Turtle-graphics tänker man sig att man har en liten sköldpadda som kryper omkring på ett papper, och lämnar ett spår efter sig. Då räcker det med kommandona (`walk <length>`), (`turn <angle>`) och

i vissa fall (`pen <up/down>`) för att kunna rita vilken graf som helst.

Vill man variera snöflingekurvan, kan man utgå från en godtycklig polygon, och byta ut mittersta tredjedelen av varje linje med en sådan polygon. Och man behöver inte heller låta omvägarna sticka utåt från centrum...



**Beware the
Program BUG!**

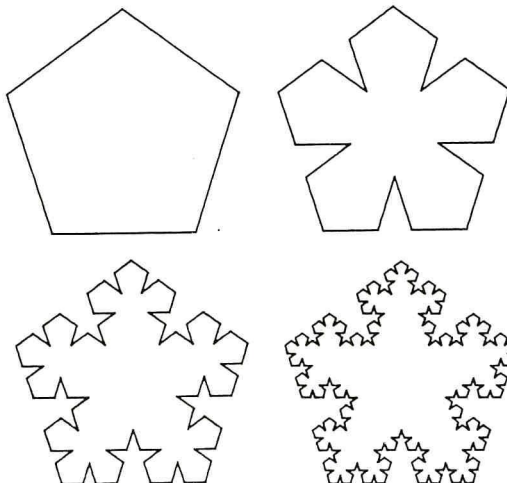
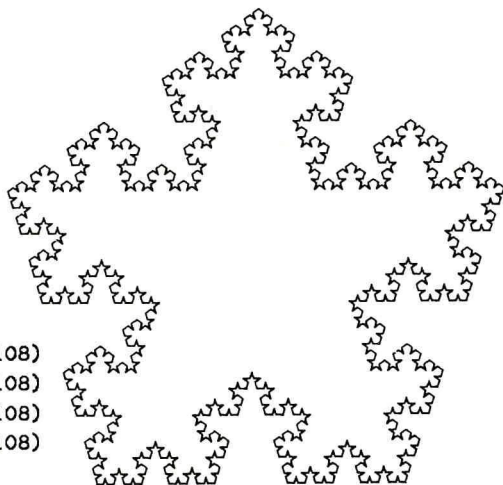


En annan vacker variant på snöflingan är *Tudor-rosen* eller *Tudor-kurvan*. Här utgår vi från en liksidig pentagon, och sticker sedan in en smal kil i mitten på varje sida så att vi får fem mindre pentagoner, en i varje hörn. Knepet är här att få rätt form och storlek på den instuckna kilen. Så här ser algoritmen ut i LISP:

```
(defun tudor (depth sidelength)
  (beginplot)
  (drawpart depth sidelength) (turn 108)
  (drawpart depth sidelength) (turn 108)
  (drawpart depth sidelength) (turn 108)
  (drawpart depth sidelength) (turn 108)
  (drawpart depth sidelength)
  (endplot))
```

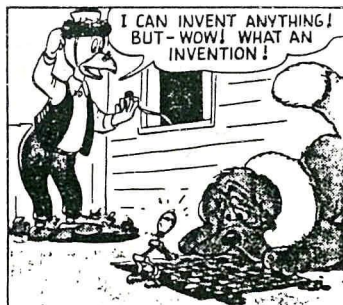
```
(defun drawpart (depth length)
  (setq length (times 2.0 (plus 1.0 (sin (quotient pi 10.0))))))
  (cond ((zerop drawpart) (walk lenght))
        (t (drawpart (sub1 depth) length) (turn 72)
           (drawpart (sub1 depth) length) (turn -324)
           (drawpart (sub1 depth) length) (turn 72)
           (drawpart (sub1 depth) length))))
```

Och så här ser den ut:

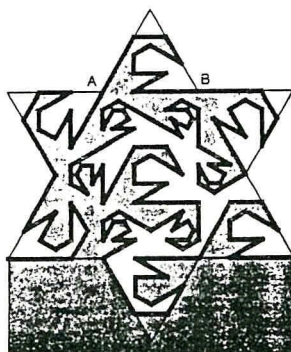
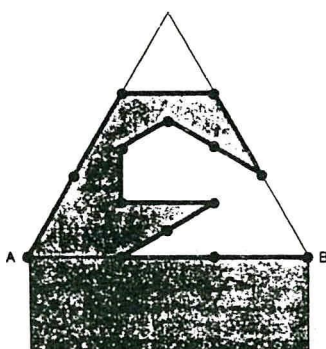


Rekursiv kanon med variationer

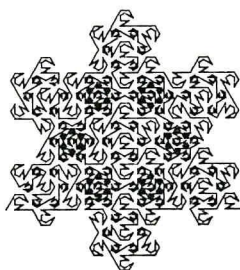
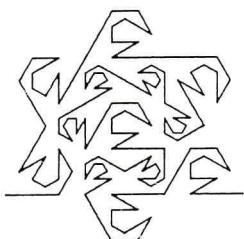
Ett modernt bidrag till matematiken om rekursiva kurvor är Mandelbrots *Peano-snowflake-curve*, som publicerades för första gången på omslaget av Scientific American i april 1978. Genom att dela in en rät linje i 13 olika långa delar på ett speciellt sätt, får man en kurva som kommer att fylla en yta begränsad av en snöflingekurva!



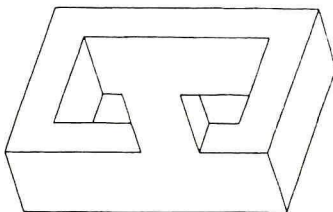
De två första stegen i konstruktionen av Mandelbrots PS-kurva:



Så här ser den ut i de tre första rekursionsgraderna:



Ochi LISP:



```
(defun other (hand)
  (cond ((equal hand 'left) 'right)
        ((equal hand 'right) 'left)))

(defun turn-to-left (hand angle)
  (cond ((equal hand 'left) (turn-left angle))
        ((equal hand 'right) (turn-right angle))))

(defun turn-to-right (hand angle) (turn-to-left hand (minus angle)))

(defun shorter (size) (quotient size (times 2.0 (cos (quotient pi 6.0)))))

(defun draw-peano-snowflake-curve (depth size)
  (setupbase)
  (beginplot)
  (setq turtle-direction 0.0
        turtle-x -0.8
        turtle-y -0.5)
  (moveto turtle-x turtle-y)
  (draw-ps-part depth 'left size)
  (endplot))

(defun draw-ps-part (depth turnside size)
  (cond ((zerop depth) (walk size))
        (t
         (setq depth (sub1 depth)
               size (quotient size 3.0))
         (turn-to-left turnside 60.)
         (draw-ps-part depth (other turnside) size)
         (draw-ps-part depth turnside size)
         (turn-to-right turnside 60.)
         (draw-ps-part depth turnside size)
         (turn-to-right turnside 60.)
         (draw-ps-part depth turnside size)
         (turn-to-right turnside 150.)
         (draw-ps-part depth turnside (shorter size))
         (draw-ps-part depth (other turnside) (shorter size))
         (turn-to-left turnside 60.)
         (draw-ps-part depth (other turnside) (shorter size))
         (turn-to-left turnside 60.)
         (draw-ps-part depth (other turnside) (shorter size))
         (turn-to-left turnside 90.))
```

```
(draw-ps-part depth turnside size)
(turn-to-right turnside 150.)
(draw-ps-part depth turnside (shorter size))
(draw-ps-part depth (other turnside) (shorter size))
(turn-to-left turnside 150.)
(draw-ps-part depth (other turnside) size)
(draw-ps-part depth turnside size))))
```

Prova själv!

Jag hoppas att de här exemplen har gjort dig nyfiken på de rekursiva kurvornas underbara värld. Med lite kunskaper i plan geometri och något bra programmeringsspråk som tillåter rekursion, samt tillgång till en grafisk skärm eller plotter kan man ägna många trevliga stunder åt de rekursiva kurvornas teori och praktik. Det är inte heller

svårt att finna en massa roliga varianter på kurvorna i denna artikel. Jag har antytt några varianter, och du kan säkert komma på fler. Och varför begränsa sig till bara två koordinater...? Börja göra en snöflinga, och med lite fantasi slutar du kanske med en ny matematisk upptäckt!

Per Lindberg



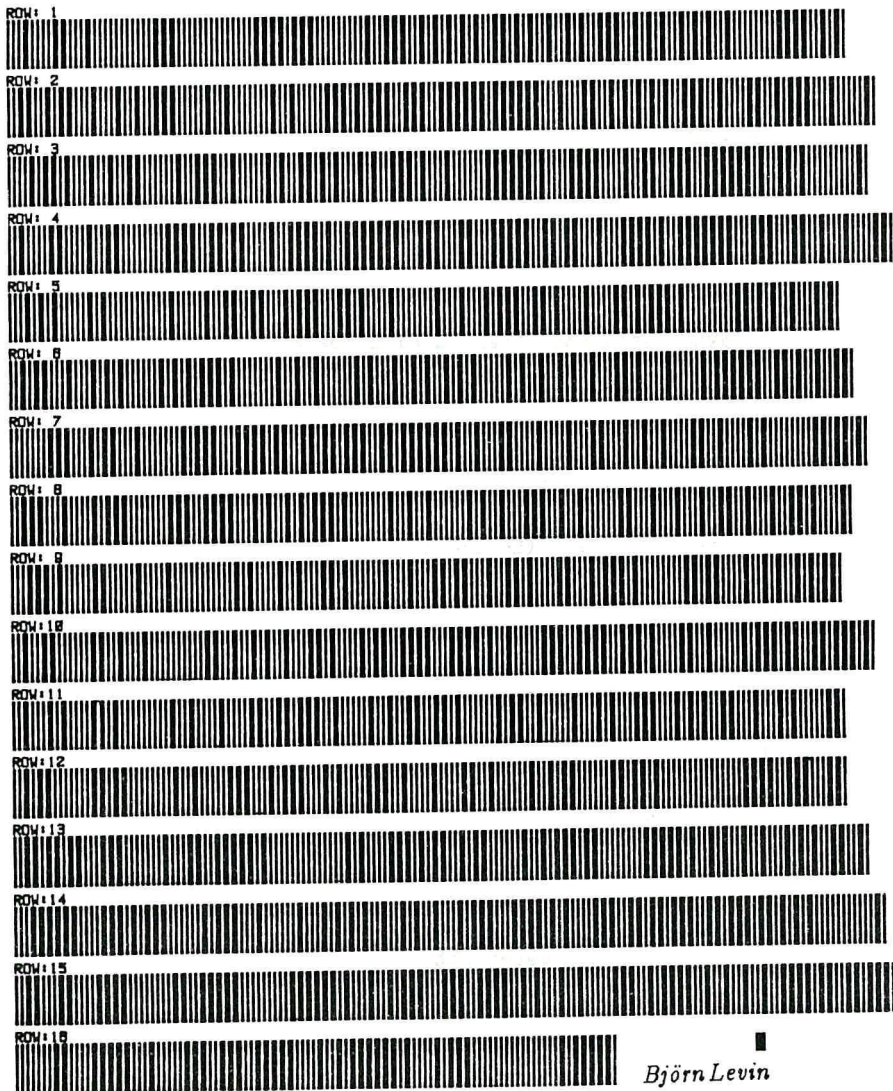
Referenser

- [1] JAMES JONES "Snowflake Plotting in ALGOL and BASIC" Creative Computing nov/dec 1978, p 146-149
- [2] M. BEELER, R. W. GOSPER, R. SCHROEPPEL "HAKMEM, Item 135" MIT Artificial Intelligence Memo No. 239, 29 Feb 1972
- [3] MARTIN GARDNER "Mathematical Games: White and brown music, fractal curves and one-over-f fluctuations" Scientific American, April 1978
- [4] BENOIT MANDELBROT "Fractals: Form, change and Dimension" W. H. Freeman and Company 1977
- [5] NIKOLAUS WIRTH "Algorithms + Data structures = Programs" p. 130-137, Prentice-Hall, Inc.
- [6] DOUGLAS R. HOFSTADTER "Gödel, Escher, Bach: an Eternal Golden Braid" p. 138-152 Basic Books Inc., New York
- [7] M. C. ESCHER "Fish and Scales" träsnitt, 1959
- [8] PATRICK HENRY WINSTON, B. K. P. HORN "LISP" Addison-Wesley Publishing company 1981
- [9] "Space-filling Curves, or How to Waste Time With a Plotter" Software-Practice and Experience Vol. 1 p. 403-410 1971

STRECKKOD TILL HP-41

Nej tyvärr, tidningen har ännu inte börjat skrivas i streckkod. Så direktinmatning till hjärnan får tyvärr vänta.

Det som finns i koden här nedanför, är ett program för numerisk integrering med Rombergs metod. Skriv ett program som beräknar funktionsvärden och kör sedan "NUMINT".



Rita en cirkel rekursivt

Ett exempel (i Algol-68) som visar att rekursiv programmering gör livet lättare. Man slipper använda några trigonometriska funktioner, men å andra sidan måste man dra kvadratroten en gång för varje rekursion.

BEGIN

```
PROC arcus = (REAL x0, y0, x1, y1, x2, y2) VOID:(
  # Rutin som ritar en cirkelbåge från (x1, y1) till
  (x2, y2) med medelpunkt i (x0, y0). Iden är att
  rita cirkelbågar genom att först rita från mitten
  till ena ändpunkten och sedan från mitten till
  andra ändpunkten. Båda dessa nya cirkelbågar ritas
  på samma sätt. Rekursionen avbryts när
  medelpunktsvinkeln är "lagom" liten. #

  REAL x1d = x1 - x0, y1d = y1 - y0,
        x2d = x2 - x0, y2d = y2 - y0

  ; PROC arc = (REAL x1, y1, x2, y2, cosang) VOID:
    # Lokal rutin som ritar en cirkelbåge från (x1,
    y1) till (x2, y2) med cos(medelpunktsvinkeln) =
    cosang #

    IF
      cosang > 0.9998 # rekursera tills vinkeln blir liten #
    THEN
      lift pen; plot (ENTIER (x1 + x0), ENTIER (y1 + y0))
      ; plot (ENTIER (x2 + x0), ENTIER (y2 + y0))
    ELSE
      REAL fudge = sqrt ((1.0 + cosang) / 2.0)
      ; REAL x = (x1 + x2) / 2.0 / fudge,
        y = (y1 + y2) / 2.0 / fudge
      ; arc (x1, y1, x, y, fudge); arc (x, y, x2, y2, fudge)
    FI
    # Börja rekursionen! #
    ; arc (x1d, y1d, x2d, y2d,
      (x1d * x2d + y1d * y2d) / (x1d * x1d + y1d * y1d))
  )

PROC circle = (REAL x0, y0, r) VOID:(
  # Nu ritar vi cirkeln genom att
  rita 4 90-graders cirkelbågar. #

  REAL x1, y1, x2, y2
  ; x1 := x0 + r; y1 := y0
```

```
; x2 := x0; y2 := y0 + r
; arcus (x0, y0, x1, y1, x2, y2)
; x1 := x2; y1 := y2
; x2 := x0 - r; y2 := y0
; arcus (x0, y0, x1, y1, x2, y2)
; x1 := x2; y1 := y2
; x2 := x0; y2 := y0 - r
; arcus (x0, y0, x1, y1, x2, y2)
; x1 := x2; y1 := y2
; x2 := x0 + r; y2 := y0
; arcus (x0, y0, x1, y1, x2, y2)
)
```

```
# Rita en cirkel med radien 10 och med origo i (25, 30) #
; circle (25, 30, 10)
```

END

Vad som fattas i detta program är själva plotrutinerna, men de är ju inte så intressanta, och dessutom blir dom så maskinberoende att det är meningslöst att ha med dom här.

Per Danielsson



SOFTNET

— data på radio

Vad skulle du tycka om att sitta hemma, med en terminal, och köra i 38400 baud på polarns dator på andra sidan stan?

Omöjligt?

— Ja, åtminstone om du skall använda televerkets nät (för du vill givetvis köra gratis!).

— Nej! Det är fullt möjligt, om du bygger en SOFTNET-nod.

SOFTNET är ett projekt vid Linköpings universitet som går ut på att konstruera metoder och hårdvara för ett amatördatanät. Nätet skall vara ickehierarkiskt och ha stor flexibilitet, för att passa datoramatörernas miljö av experimenterande och nytänkande.

SOFTNET är uppbyggt av ett godtyckligt antal likadana *noder*. En nod består av en radiosändare/mottagare på 432MHz 10W, och en styrdator med två 6809-processorer.

Noderna kommunicerar med varandra med s.k. *packet switching*, där dataströmmen delas upp i små paket, om ca 100 bytes vardera, som vart och ett tar sig fram på lämpliga vägar genom nätet. överföringshastigheten mellan två noder ligger på 100Kbaud, men den totala hastigheten beror på hur många som samtidigt använder nätet.

SOFTNET är ett passande namn, eftersom modernas funktion långtifrån är hårdlörd i hårdvaran. Noden programmeras i högnivåspråket FORTH, som med sin hierarkiska struktur passar bra i multianvändarmiljö. Varje nod kan programmeras från andra noder *över radio!*

Att SOFTNET är ickehierarkiskt innebär i detta fall att ingen del av nätet

är överställd någon annan. Detta är viktigt för att hela nätets funktion inte skall skadas av att en enstaka nod faller ur. När en ny nod startas, anpassar sig nätet automatiskt till detta, och kan direkt börja skicka paket via den.

Den höga överföringshastigheten, och enkelheten varmed man skickar ett par bytes till en annan nod gör nätet lämpligt t.ex. för interaktiv fjärrkommunikation i full duplex mellan en terminal och en dator, eller för styrningsändamål.

Routing-algoritmer

En av uppgifterna för varje nods styrprogram är att hitta bra vägar att skicka paket för att dessa skall nå sitt slutmål så fort som möjligt. Det har gjorts en hel del teoretiskt arbete på att få dessa s.k. routing-algoritmer effektiva, och säkra.

Betrakta nätet i fig. 1. Antag att nod A vill sända ett paket till C. Den kortaste vägen går givetvis över F. Det är lätt att programmera noderna för att klara överföringar i detta nät.

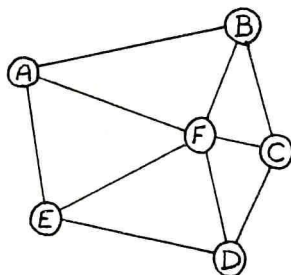


Fig. 1

I fig. 2, däremot är den geografiskt kortaste vägen inte längre möjlig på grund av ett för radiovågor ogenomträngligt hinder.

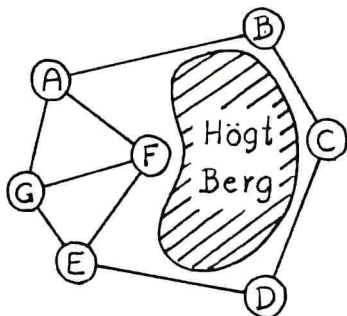


Fig. 2

Denna situation är ett exempel på de problem som kan uppkomma när routing-algoritmer skall konstrueras. I SOFTNET har dessa problem lösts med hjälp av metoder från ett helt annat område, nämligen datoranalys av bilder. Man utnyttjar här en metod som används vid kantdetektering i rasterbilder, som i fig. 2 gör det möjligt för ett paket att följa "kanten" runt berget fram till C.

FORTH ger flexibilitet

Att använda FORTH innebär inte bara att man får ett enkelt och koncist språk för att ge *kommandon* till andra noder. Man har dessutom utvecklat en speciell multiprocess-FORTH för att använda i noderna.

En nod kan även lagra och kompilera FORTH-procedurer i avlägsna noder för att klara av saker som annars skulle kräva överföring av mycket större datamängder.

Säg exempelvis att nod B i fig. 1 ett tag framöver kommer att behöva skicka samma paket till både nod C och D. B antas nu veta att både C och D nås bäst via

nod F, och skickar då följande paket till F:

```
:SPLIT DUP C SEND D SEND;
```

Detta innebär att det i F lagras en ny procedur SPLIT, vars uppgift är att duplicera ett paket, och sända varsitt exemplar till C och D. När B sedan vill sända paket till noderna, skickar den paketet

```
...data... SPLIT
```

till F, där SPLIT gör resten av jobbet.

Intressanta algoritmer

Det finns algoritmer som i denna typ av nätverk kan användas av varje nod för att automatiskt skaffa sig en bild av alla kommunikationsvägar i nätet. Man behöver alltså inte i förväg tala om för sin nod var övriga noder finns.

En annan algoritm kan användas för att skicka ett paket så att det besöker varje nod exakt en gång för att t.ex. samla statistik om nätet, för att slutligen återvända till utgångsnoden med "resultatet".

Algoritmer för nätverkskommunikation är ett inte på något sätt uttömt forskningsområde. Många upptäckter kommer säkert att kunna göra kommunikationen i denna typ av datornät effektivare och säkrare.

Hårdvaran

SOFTNET är inte bara en idé. Redan för ett halvår sedan byggde man i Linköping en testprototyp till en nod, och två till är på väg.

En av målsättningarna med systemet är att en SOFTNET-nod inte skall behöva kosta mer än vad andra periferienheter i ett vanligt mikrodatorsystem kostar. För ca 5000kr bör man alltså kunna bygga sig

en nod, koppla in antenn och terminal (eller egen dator) och ge sig ut i etern.

Linköpings prototyp består av 2st 6809-processorer, 32Kbyte RAM, 8Kbyte PROM, lite kringkretsar och en sändare/mottagare för 432MHz med uteffekt 10W. Den ena CPU:n styr sändaren och mottagaren, och den andra sköter anpassningen till användaren och kör routing-algoritmer. Som antenn duger en stavantenn ca 30cm lång vilket kan ge en räckvidd på uppemot 5mil.

Vad tycker televerket?

Som med alla grejor som har med radiokommunikation att göra måste televerket även här vara med i leken. Det är främst två egenskaper hos SOFTNET som skulle kunna göra televerket tveksamma. Det ena är att SOFTNET-sändaren fjärrstyrs över radio, det andra att en SOFTNET-nod vidarebefodrar meddelanden åt andra. Båda dessa saker är i princip förbjudna för radioamatörer.

Men televerket är ändå positivt inställda till SOFTNET. Förbundet Sveriges sändareamatörer, SSA, har blivit konsulterade av televerket. De säger sig också vara för SOFTNET. Slutresultatet blir nog att SOFTNET får tillstånd, på villkor att driften administreras av den ursprungliga projektgruppen i Linköping. Detta är nog också nödvändigt för att nätet skall fungera bra och säkert.

En sak till, för att få använda nodens sändare måste man ha ett amatörradiocertifikat. Det räcker här med ett s.k. T-certifikat, som inte kräver några kunskaper i morse-telegrafering (–och tur är det, kan jag intyga!).

Dag Rende

Litteratur och synpunkter

Konstruktörerna är tacksamma om de får synpunkter och idéer från alla som är intresserade av SOFTNET. De skickar dessutom gärna de forskningsrapporter som beskriver SOFTNET. Snart kan de även skicka kopplingscheman på deras nod, så att man själv kan börja bygga.

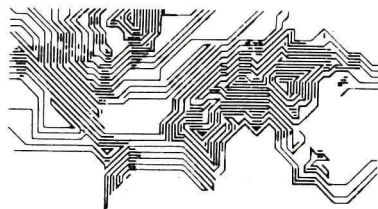
Skriv eller ring till:

Jens Zander eller Robert Forchheimer
Linköpings tekniska högskola
Institutionen för Systemteknik
581 83 Linköping

tel: 013-111700 (växel)

Litteratur

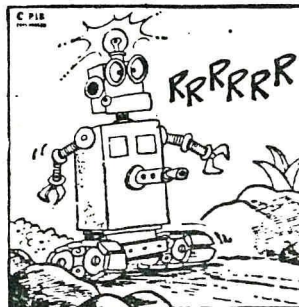
- [1] ZANDER J, FORCHHEIMER R
"Preliminary Specifications for a Distributed Packet Radio Network for Computer and Radio Amateurs" Internal Report, LiTH-ISY-I-0424, January 1980
- [2] PERSSON I, FORCHHEIMER R
"Design Considerations of a Distributed Packet Radio Network using the Amateur Radio bands" Internal Report, LiTH-ISY-I-0408, May 1980
- [3] QVIGSTADF, MATTS S
"Construction of a packet radio node control computer" Internal report, LiTH-ISY-I-0491, December 1981
- [4] ZANDER J
"M-FORTH, Multitasking FORTH Implementation for the 6809"



IBM 370 Features and Op-Codes

Early Card Lace
 Circulate Memory
 Branch on Burned-out Indicator
 Branch on Blinking Indicator
 Branch and Hang
 Branch on Chip Box Full
 Branch on Power Off
 Branch on Sleepy Opreator
 Inquire and Ignore
 Reverse Parity and Branch
 Branch on Bug
 Read and Write While Ripping Tape
 Add Improper
 Divide and Overflow
 Substract and Reset to Zero
 Add and Reset to Zero
 Scramble Program Status Word
 Punch Invalid
 Read Card and Scramble Data
 Select Stacker and Jam
 Read Invalid
 Rewind Card Reader
 Backspace Card Reader
 Read Print and Blush
 Forms Skip and Runaway
 Stacker Select Disc
 Write Wrong-length-record
 Write Noise Record
 Seek Record and Scar Disc
 Eject Disc
 Rewind Disc
 Backspace Disc
 Punch Disc
 Punch Operator
 Execute Invalid Op Code
 Feed Card and Jam
 Read Chaos (Special Feature)
 Sharpen Light Pencil
 Transfer and Drop Bits
 Erase Card Punch
 Read Inter-record Gap
 Read Noise Record
 Erase Read Only Storage
 Destroy Storage Protect Key
 Update and Erase Record
 Move and Drop Bits
 Branch and Blow Up
 Print On Fly
 Move and Wrap Core
 Move Continous
 Develop Ineffective Address
 Halt and Catch Fire - Privileged Op
 Scatter Print
 Reinitailize Meter
 Update Transaction
 Reduce Thruput

Print and Break Chain
 Lose Message and Branch
 Burst Selector Channel
 Invert Record and Branch
 Illogical Or
 Illogical And
 Bite Baudy Bit and Branch
 Triple-Pack Decimal
 Slip Disc
 Stacker Upset
 Uncouple CPU's and Branch
 Scramble Channels
 Rewind and Break Tape
 Make Tape Invalid
 Reverse Drum Immediate
 Transfer and Lose Return
 Print and Smear
 Create Machine Bug
 Loop Continous
 Branch on Index Missing
 Execute Operator
 Ignore Supervisor Call
 Write Count Key and Garbage
 Read Count Key and Garbage
 Electrocute DP Manager and Branch
 Create Data
 Random Bug Generator
 Hop Skip and Jump
 Jump on Flag and Eat it
 Branch and Loop Indefinate
 Branch on CE Ground
 Trap Secretary and Halt
 Byte and Run
 Gulp and Store Bytes
 Burp and Clear Bytes
 Eat Card
 Execute Supervisor
 Random Print



PARTIALEVALUERING

eller

HUR MAN FÅR EN KOMPILATOR FÖR (NÄSTAN) INGENTING

En intressant sorts program är en partialevaluator. En dylik tar ett annat program och vissa kända indata till detta program och skapar en "specialiserad" version av programmet för dessa indata. Det specialiserade programmet gör samma sak som det ursprungliga, men de kända indata är inbyggda i det. Tag till exempel följande PASCAL-funktion:

```
function foo(x,y:integer):integer;
begin
  foo := x+y
end;
```

Om vi partialevaluerar detta program under förutsättningen att parametern y är 1, så får vi följande nya program:

```
function fool(x:integer):integer;
begin
  fool := x+1
end;
```

Fool är alltså en specialiserad variant av foo, den gör samma sak som foo gör när parametern y är 1, d.v.s. $fool(x) = foo(x, 1)$. På liknande sätt kan vi partialevaluera foo under andra förutsättningar.

I princip så tar alltså en partialevaluator och byter ut alla förekomster av en eller flera variabler mot ett förutbestämt värde och förenklar koden så mycket det går.¹

Riktigt intressant blir partialevalueringen först när programmet innehåller villkor av olika slag. Man kan då få en möjlighet att ta bort onödiga tester och därmed förenkla programmet. Betrakta t.ex. denna funktion:

```
function bar(x,y:integer):integer;
begin
  if x < 0 then
    bar := y + 1
  else
    bar := y - 1
end;
```

Om vi partialevaluerar bar under förutsättningen $x = -1$, så får vi:

```
function bar1(y:integer):integer;
begin
  bar := y - 1
end;
```

vilket är avsevärt enklare. Eftersom villkoret $x < 0$ alltid är sant om $x = -1$, så kan vi byta ut hela if-satsen mot dess then-gren.

En annan förenklingsteknik som används är att "öppna" funktioner. Detta innebär att funktionsanrop byts ut mot definitionen av en anropade funktionen, förenklad så långt som möjligt. Så kan uttrycket $5 * bar(-1, z)$ partialevalueras till $5 * (z - 1)$.

Ursprungligen så framställdes partialevaluators för att skraddarsy generella program. Man kunde till exempel ha ett jättelikt program som uppfyllde alla ens önskningsar, men som var för stort och tungt för att kunna användas i praktiken. Ifall man bara skulle använda programmet för vissa ändamål kunde man med en partialevaluering minska ned programmets storlek genom att det inte längre behövde hantera olika värden på paramstrar som man visste alltid skulle ha samma värde i det sammanhang man ville använda programmet.

Nu när läsaren har grepp på vad partialevaluering går ut på så skall jag ta upp hur man kan använda denna teknik för att, till synes magiskt, framställa kompilatorer.

Kompilatorgenerering

För att resonemanget skall bli hanterligt måste jag införa lite matematisk notation enligt följande:

- Program betraktas som funktioner, vars parametrar är indata och värde utdata. Påståendet $f(IN) = UT$ innebär alltså att programmet f ger utdata UT när det matas med indata IN .
- Program skrivna i maskinkod för vår tänkta maskin skrivs med små bokstäver (som " f " ovan).²
- Allt som inte är maskinkod skrivs med stora bokstäver (som " IN " ovan).

Exemplet med partialevaluering av funktionen foo kan beskrivas på följande sätt i vår notation:

Från början hade vi

$$foo(X, Y) = X + Y$$

Om vi kallar partialevaluatorn för pe och låter den ha två parametrar, den funktion som skall partialevalueras och de av funktionens parametrar som är fixa, så gäller:

$$pe(foo, Y) = foo1,$$

där

$$foo1(X) = foo(X, Y) = X + Y$$

Sätter vi $Y = 1$ så får vi samma exempel som ovan:

$$pe(foo, 1) = foo1,$$

där

$$foo1(X) = foo(X, 1) = X + 1.$$

Antag nu att vi vill implementera ett nytt språk på vår maskin, t.ex. språket BABEL. Det enklaste sättet att göra detta är att skriva en interpretator för BABEL. Interpretatorer är oftast ganska små och okomplicerade program (i alla fall om man jämför med en kompilator).

Låt oss kalla interpretatorn för $eval$. $Eval$ vill ha två saker för att kunna exekvera ett BABEL-program: själva programmet och dess indata. Vi kan alltså skriva:

$$eval(F, IN) = UT$$

Här har vi kallat programmet F . Observera att vi använder $STORA F$, eftersom BABEL-program inte är i vår maskins maskinkod.

Antag att vi hade en kompilator för BABEL. Låt oss kalla den $compile$. Då skulle det gälla att:

$$compile(F) = f$$

där f är ett maskinkodsprogram som utför samma beräkning som BABEL-programmet F , dvs

$$f(IN) = UT.$$

Vad skulle ett uttryck som $pe(eval, F)$ betyda? Här har vi partialevaluerat interpretatorn med avseende på ett speciellt BABEL-program, nämligen F . Enligt vad vi vet om partialevaluering så skall vi som resultat få ett program (i maskinkod, märk väl, eftersom det är en specialisering av $eval$, vilken är ett maskinkodsprogram) med en parameter IN , som alltid ger samma värde som uttrycket $eval(F, IN)$ med andra ord, om vi kallar $pe(eval, F)$ för g , så får vi:

$$g(IN) = UT$$

Men detta är precis samma funktion som det kompilerade BABEL-programmet *f* ovan. Med andra ord: $pe(eval, F)$ har kompilerat BABEL-programmet *F* till maskinkod!

Stanna upp och begrunda. Vi har utgått ifrån en partialevaluator – som är ett relativt enkelt program och som kan skrivas en gång för alla – och ifrån en interpretator – som också är förhållandevis enkel och rakt på sak att skriva – och fått en kompilator.

Varför det fungerar

En intuitiv förklaring till varför detta fungerar kan man ge så här: När *eval* interpreterar ett BABEL-program, så måste den alltid utföra precis de maskininstruktioner som behövs för att utföra den beräkning som BABEL-programmet beskriver. Dessutom måste interpretatorn utföra en mängd andra maskininstruktioner för att avkoda BABEL-programmet, sköta symboltabell och annat som interpretatorer gör. Det är dessa andra instruktioner som gör att interpretering är så långsam. Nu var det ju sagt att en partialevaluator kunde eliminera sådan kod som var onödig, om man i förväg kände till en del av indata. Men BABEL-programmet är ju indata till interpretatorn, så genom att partialevaluera så får man bort allting som har med avkodning av programmet etc. att göra – har man bara ett program så finns det ju bara ett sätt att avkoda på – och det som till slut blir kvar är precis de maskininstruktioner som utför BABEL-programmets beräkning, det vill säga, BABEL-programmet har blivit översatt (kompilerat) till maskinkod.

När man svalt detta så kan man kanske argumentera så, att visserligen

har vi fått en kompilator, men för att göra en kompilering så måste man partialevaluera *hela* interpretatorn varje gång, vilket ju måste ta en väldig tid. Detta är en alldeles riktig invändning, så därför bildar vi följande uttryck:

$$pe(pe, eval) = h$$

Vi låter alltså partialevaluatorn partialevaluera sig själv med avseende på BABEL-interpretatorn. Jag skall inte försöka förklara vad som egentligen händer här utan bara nöja mig med att tala om att programmet *h*, som blir resultatet, är identisk med BABEL-kompilatorn *compile* som vi talade om tidigare, men inte ville skriva. Det gäller alltså att:

$$h(F) = compile(F) = f.$$

För att gå ytterligare ett steg kan vi bilda följande fantastiska uttryck:

$$pe(pe, pe) = gen \quad (!)$$

Vad kan då detta vara? Jo, tro det eller ej – en kompilatorgenerator! $gen(eval) = compile$, man tar en interpretator och ut får man en kompilator.

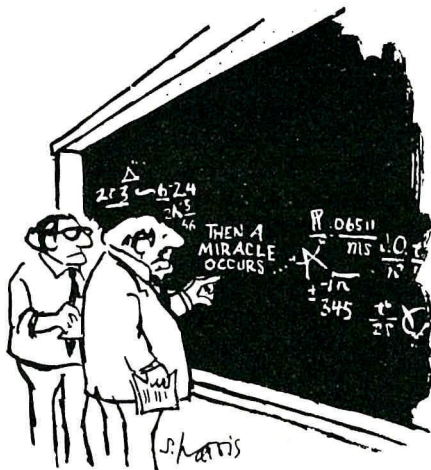
Pågående forskning

En sak man undrar är varför inte detta system att framställa kompilatorer på är mera spritt, det förefaller ju underbart att man – bara genom att skriva en interpretator – kan få en kompilator på köpet. (Partialevaluators behöver man ju bara skriva *en* gång för alla). Detta beror huvudsakligen på två saker: dels är metoden ganska okänd och dels är existerande partialevaluators för komplicerade för

att kunna partialevaluera sig själva, vilket är nödvändigt för att man skall få en effektiv kompilator.

Forskning på området pågår dock, faktiskt här i Sverige. Linköping ligger bra framme, men även i Uppsala arbetas det på problemen. I Linköping är det bland annat **Anders Haraldsson** och **Per Emanuelsson** som arbetar med partialevaluatorer (båda har doktorerat på ämnet). I Uppsala arbetar **Kenneth Kahn** på att implementera PROLOG på LISP-maskiner, genom att skriva en PROLOG-interpretator i LISP och en partialevaluator för LISP skriven i PROLOG. Tillvägagångssättet för hans del blir i princip samma som ovan men något flera steg krävs, eftersom partialevaluatorn själv är skriven i PROLOG och inte i LISP.

Lars-Henrik Eriksson

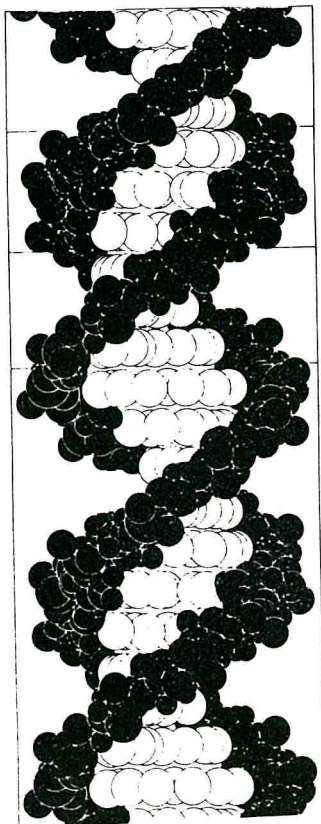


"I THINK YOU SHOULD BE MORE EXPLICIT HERE IN STEP TWO."

Fotnötter

1. Här förenklar jag en aning, man kan också partialevaluera program under förutsättningen att parametrarna har godtyckliga egenskaper, t.ex. att X är större än 0. I sådana fall måste man fortfarande behålla parametern X , men det finns ändå möjlighet att förenkla programmet. Att anta att en parameter har ett bestämt värde är bara ett specialfall - då har parametern egenskapen att alltid ha detta bestämda värde. I denna artikel kommer vi emellertid bara att partialevaluera med avseende på bestämda parametervärden.

2. Vad vår maskin och maskinkod egentligen är spelar ingen roll för diskussionen, men i praktiken så är det antagligen LISP eller liknande. Att använda en PASCAL-maskin med P-kod som maskinspråk (t.ex.) är inte möjligt eftersom programmen måste kunna behandla andra maskinkodsprogram. De enda språk där detta är möjligt är assembler, APL och LISP-liknande språk, och assembler vill man inte använda.



SAIL

Stanford Artificial Intelligence Language

SAIL är ett tämligen kraftfullt ALGOL-språk, som har utvecklats vid Stanford University. Sail har bland annat Macros som expanderas under kompileringen, något som de allra flesta språk saknar, och som kan vara mycket användbart. Men det kanske roligaste med SAIL är den associativa Databasen! Något som annars knappast finns annat än i t.ex. PROLOG. Dessutom har SAIL datatyperna LIST (Listor), SET (Mängder) och den speciella datatypen ITEM. Ett ITEM är inte någon variabel utan snarare en konstant, som man kan hänga på en massa egenskaper. SAIL har också en mycket stark process-hantering (både seriella och time-sharade). Man kan ha procedur-variabler i SAIL, och sedan anropa dem med funktionen APPLY (Hej, alla LISP-freaks!), som tar ett procedur-namn eller en procedur-variabel och en argumentlista som argument. En annan rolig sak med SAIL är att man kan "defaulta" parametrar till en procedur, och sedan utelämnna just de parametrarna när man anropar proceduren, som då antar det värde som man har "defaultat" dem till. Dokumentation på SAIL finns (tror jag) att köpa på QZ ("A beginners guide to SAIL"), om man inte har konto på ODEN förstås, för där finns all SAIL-dokumentation på DOC:-arean (någon elak person har stadat bort den från NADJA:s). Nedan följer ett litet exempel på ett SAIL-program som utnyttjar datatyperna SET och ITEMS, samt lite Macros.

Den break-table jag skriver om nedan är en slags tabell som innehåller brytkaraktärer, som används av funktionen Scan. Scan returnerar substrängen till en av brytkaraktärerna som man specificerat i SetBreak hittas I proceduren Rotate!list så är sista parametern defaultad till 1, d.v.s. att den blir 1 om man inte specificerar annat. CONTINUE är en mycket trevlig sats! Med den behöver man inte någon flag och/eller goto - programmering. De strängar som finns efter BEGIN & END måste matcha med det BEGIN/END som omsluter blocket. Ett exempel på körning:

This is a Sail program!

Yes ?qno

Father of QNO is ALLAN.

QNO's grandfather is SUNE.

QNO is father of FOO.

And grandfather of FOOBAR.

And grandfather of BARFOO.

QNO is father of BAR.

And grandfather of BARF.

No more relations.

Yes ?sune

SUNE is father of SVEN.

And grandfather of HARRY.

SUNE is father of ALLAN.

And grandfather of QNO.

And grandfather of SVANTE.

No more relations.

Yes ?svante

Father of SVANTE is ALLAN.

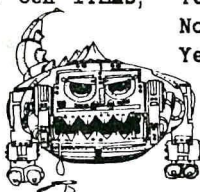
SVANTE's grandfather is SUNE.

No more relations.

Yes ?quno

No such person!

Yes ?!C



(CONS 'Björn 'Carlsson)

BEGIN "Example of Sail"

```
" En sträng skriven så här mitt i programmet är en kommentar!"
" Vi måste ha PNames till våra ITEMS..."
REQUIRE PNames;
" Definiera några Macros, som kan vara bra att ha."
DEFINE ! = "COMMENT",
    CrLf = "(13&10)",
    FOREVER = "WHILE TRUE",
    OF = "XOR",
    IS = "EQV",
    Relation(Sort,X,Y) = "MAKE Sort OF Y IS X";
```

```
STRING user!line; ! Man får ha "!" och "$" i variabel namn. ;
ITEM father, sune, allan, sven, harry, svante, qno, foo, bar, barf,
    foobar, barfoo;
ITEMVAR person,x,y; ! En variabel som kan ta ett ITEM;
SET dad, farfar;
BOOLEAN Flag;
INTEGER $void$;
```

```
Print("This is a Sail program!",CrLf);
SetBreak(1," ","","ISK"); ! Definiera en "break-table" med <space> och
    komma som bryt-tecken. ;
" ... Sätt upp lite relationer ... "
Relation(father,sune,allan); Relation(father,sune,sven);
Relation(father,allan,svante); Relation(father,allan,qno);
Relation(father,qno,bar); Relation(father,qno,foo);
Relation(father,sven,harry); Relation(father,bar,barf);
Relation(father,foo,barfoo); Relation(father,foo,foobar);
```

FOREVER DO

BEGIN "Main loop"

```
IF NOT Length(user!line) THEN
    BEGIN
```

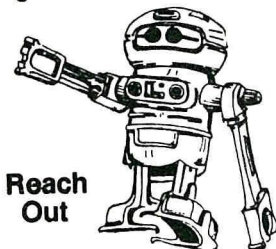
```
    Print("Yes ?");
    user!line := Inchwl
```

```
    END;
```

```
person := CVSI(Scan(user!line,1,$void$),Flag);
```

```
" Scanna strängen med 'break-table' nummer 1"
```

```
IF Flag THEN
```



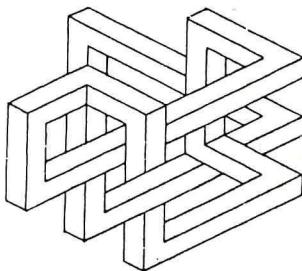
Reach
Out



```
BEGIN "Error"
  Print("No such person!",crlf);
  CONTINUE "Main loop"
END "Error";
dad := father OF person;
IF Length(dad) THEN
  BEGIN
    Print("Father of ",person," is ",Cop(dad),".",crlf);
    farfar := father OF Lop(dad);
    IF Length(farfar) THEN
      Print(person,"'s grandfather is ",Lop(farfar),".",crlf)
    END;
  FOREACH x SUCH THAT father OF x IS person DO
    BEGIN "Search relatives"
      Print(person," is father of ",x,".",crlf);
      FOREACH y SUCH THAT father OF y IS x DO
        Print(" And grandfather of ",y,".",crlf)
      END "Search relatives";
    Print("No more relations.",crlf)
  END "Main loop"
END "Example of Sail"
```

*ER DIN SWITCH,INI
NOE Å SKRYTE AV ?*

(SE HELP SWITCH OG NES 37 OG 38)



National Semiconductor 16032

NS16000 är National Semiconductors nya serie av 16-bitarsprocessorer. Kretsarna finns ännu inte att köpa och enligt återförsäljare kommer de troligen att bli tillgängliga "i slutet av 1982 eller början av 1983".

CPU NS16008 och NS16016

De två minst avancerade processorerna, NS16008 och NS16016, kommer bara att få 64Kbyte adressrymd och 16-bitars register och är därför inte särskilt intressanta.

CPU NS16032

NS16032 är den största processorn, som National Semiconductor i dagsläget planerar att tillverka. Den ska ha 16Mbyte osegmenterad virtuell adressrymd, men eftersom de generella registren och adressdelen i maskininstruktionerna har plats för 32 bitar kommer programvara, som utvecklas för den, att utan omskrivning kunna köras på eventuella framtida processorer med större adressrymd. I likhet med NS16008, NS16016 och MC68000 kommer NS16032 att vara mikroprogrammerad.

Instruktionsrepertoar

Både maskininstruktionerna och adresseringsmodernerna bär tydliga spår av att konstruktörerna har haft PASCAL i tankarna och kan närmast karaktäriseras som en korsning av PASCAL Microengine och VAX-11.

Instruktionskoderna, som kan ligga på godtycklig bytegräns, är en till tre bytes långa och följs av noll eller flera bytes med adresser och offsets.

Processorn har en buffert, där den kan läsa in upp till åtta instruktionsbytes i förväg. Om någon av dessa bytes inte går att läsa från minnet sparas status undan, så att man ska få ett avbrott när och om man försöker exekvera den oåtkomliga instruktionen.

Alla logiska och aritmetiska operationer kan operera på byte, ord och dubbelord. Ord och dubbelord kan börja på godtycklig bytegräns. Processorn har särskilda operationer för hantering av bitar, bitfält, strängar, köer, BCD-aritmetik och konvertering mellan operander av olika längder, med och utan sign extension. Någoting mycket ovanligt är att den har två instruktioner för heltalsdivision, en som avrundar neråt och en som avrundar mot noll, och motsvarande två moduloinstruktioner.

Alla instruktioner kan använda alla adresseringsmoder för alla operander, med undantag för treoperandinstruktioner för arrayindexberäkning, bitfält, matris-multiplikation och polynomberäkning, där en av operanderna alltid ligger i ett register. Strängoperationerna, som måste kunna avbrytas och återupptas var som helst, har inga explicita operander, utan hämtar sina argument från registren.

NS16032 har minnesmappad I/O.

Operativsystemsunderstöd

Processorn har två moder, användarmod och systemmod. När den körs i användarmod kan den inte utföra operationer, som påverkar minneshanteringen eller interruptsystemet. Det finns en särskild instruktion, SVC, för att anropa operativsystemet.

Felsökningshjälpmedel

Genom att lägga in instruktionen BPT får man en brytpunkt på det stället i programmet. Om man sätter T-biten i processorstatusregistret kan man få ett avbrott efter varje instruktion, så man kan exekvera en instruktion i taget.

Register

- R0-R7 är 8 generella 32-bits register.
- PC är programräknaren.
- SB innehåller startadressen för modulens statiska data.
- FP pekar på början av den exekverande procedurens stack frame.
- US är användarens stackpekare.
- IS är systemets stackpekare.
- INT BASE pekar på systemets avbrottsvektor.
- PSR innehåller processorns statusflaggor.
- MOD pekar på den exekverande modulens entry i modultabellen.

Adresseringsmoder

- Register. Operanden ligger i ett registerpar, ett register, de 16 lägsta bitarna i ett register, de 8 lägsta bitarna, i ett flyttalsregisterpar eller i ett flyttalsregister, beroende på instruktionen.
- Register indexerat. Operanden ligger på minnesadressen $(R_n) + Displacement$.
- Pusha på stacken.
- Poppa från stacken.
- Överst på stacken, eller snarare underst.
- Relativt stack pointer. Effektiva adressen är $(SP) + Displacement$.
- Relativt frame pointer.
- Relativt frame pointer indexerat. Effektivadressen beräknas som $((FP) + Displacement_1) + Displacement_2$. Relativt en relativ pekare i minnet!
- Relativt static base.

52 • Relativt static base indexerat.

- Relativt programräknaren.
- Absolut adress.
- Omedelbart efter instruktionskoden.
- Extern. Effektiva adressen beräknas ur modultabellen som $((Mod) + 4) + 4 \times Displacement_1 + Displacement_2$. Kan användas för att länka in och relokeras om separatkompileerade moduler under körning.
- Indexerat byte, ord, dubbelord och dubbeldubbelord. Effektiva adressen är 1, 2, 4 respektive $8 \times (R_n)$ plus en annan adress, som anges med någon av ovanstående adresseringsmoder.

FPU NS16081

Flyttalsprocessorn NS16081 kan utföra de fyra räknesätten, dra kvadratrötter, plocka ut heltalsdelen och annat smått och gott på flyttal i enkel och dubbel precision.

Register

- FR0-FR7 är 8 32-bitars flyttalsregister. Operationer i dubbel precision använder ett jämnt och nästa udda flyttalsregister.
- FSR är NS16081s statusregister.

MMU NS16082

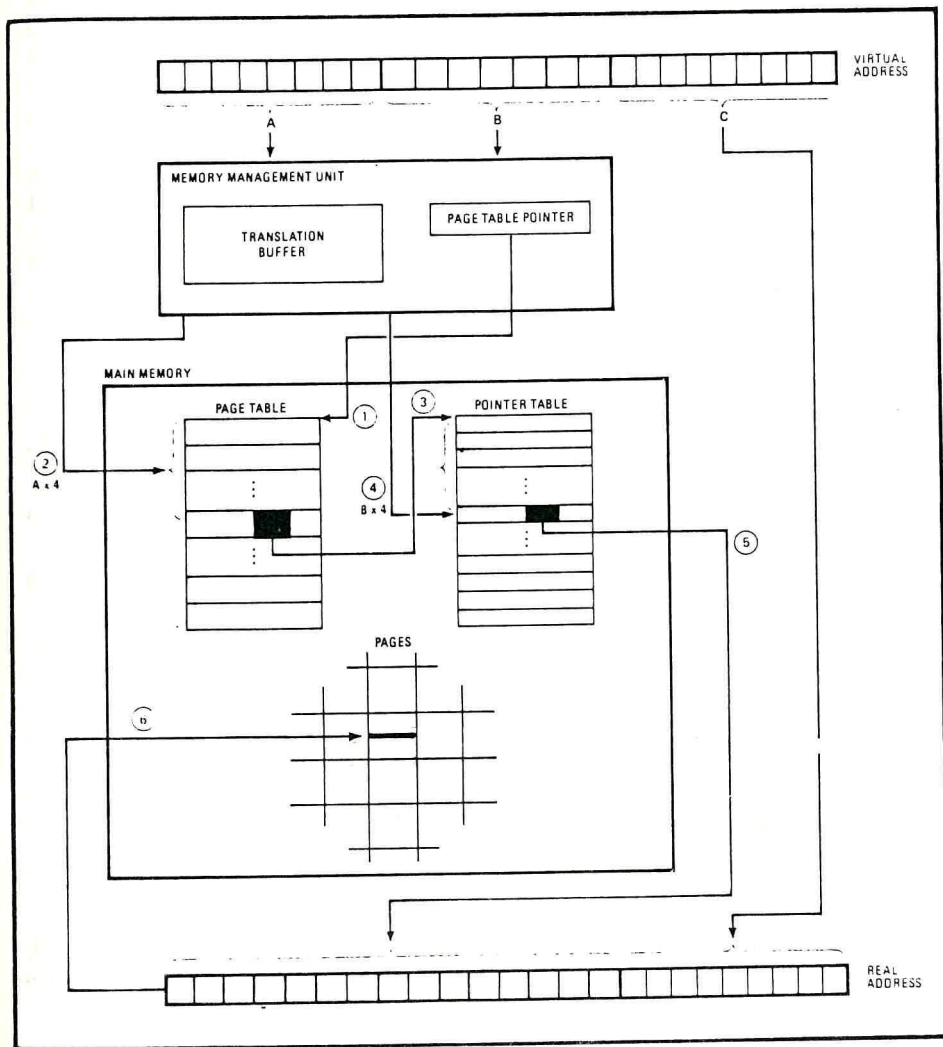
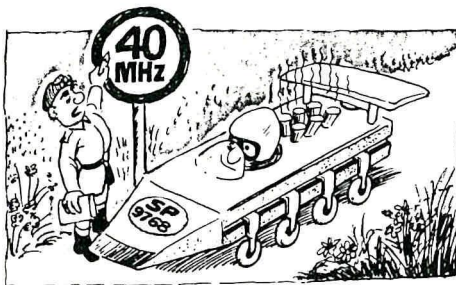
NS16082 kan relokeras minnesadresser och skydda mot otillåten läsning och skrivning i minnet. Till skillnad från Z8000s och MC68000s minnesskyddskretsar har den sina tabeller i det vanliga minnet. För att snabba upp översättningen av virtuella adresser till fysiska har NS16082 en buffert, där den behåller de 32 senast använda adressöversättningarna.

Adressöversättningen går till så här. Om adressen inte finns i översättningsbufferten använder MMU sin sidtabellpekare (1) plus de 8 första adressbitarna (A) för att indexera sig in i

sidtabellen (2). Där hittar den en pekare till nästa tabell (3), som tillsammans med de nästa 7 adressbitarna (B) pekar ut en adressöversättning (4) till fysiskt sidnummer (5).

NS16032 och NS16082 ska få fullt virtuellminnessupport.

Jan Michael Rynning



Motorola 68000

MC68000 är Motorolas 16-bitars maskin, som nu funnits ca 1,5 år. Den är, till skillnad från 8086 och Z8000, mikroprogrammerad. Detta betyder naturligtvis inte att den är bättre bara för det, men man kan misstänka att det är detta som ligger bakom den trevliga instruktionsuppsättningen.

Lite hårdvara först:

- 24-bitars ej multiplexad adressbus
- 16-bitars databuss
- Möjlighet att köra i 68000 kompatibilitetsmod (map på bussignaler)
- 7 interruptnivåer
- 200 möjliga interrupt-vektorer

8 Mhz-versionens kortaste instruktioner tar .5 μ sek, flytta ett minnesord till minnesord med hjälp av pekare tar 1.5 μ s, förutsatt att minnesaccesstiden är mindre än 220 nsek.

Processorn har inbyggt skydd för exempelvis ett OS genom möjligheten att köra i supervisor mode resp user mode. I supervisor mode kan man göra allt, t ex ställa om interrupt-nivån, medan man i user mode snällt får be OSet om tjänster via TRAP-instruktioner.

Processorn har 8 dataregister i storlek **byte**, **word** (16 bitar) eller **long word** (32 bitar) beroende på instruktionsmod. Dessutom finns 8 adressregister, varav ett är stackpointern (eller egentligen en av stackpointerna, det finns en för supervisor mode och en för user mode). Adressregisterna kan ha alltid storleken 32 bitar (men man kan ladda 16-bitars tal vilka då sign-extendas till 32-bitars tal).

Adressrymden är 24 bitar (linjär, dvs ej segmenterad), dock är alla adressregister 32 bitar (!).

Instruktionsuppsättningen är uppbyggd runt ett fåtal instruktioner, där sedan de mest använda har fått kortvarianter. Naturligtvis finns det konstiga instruktioner också, men de är i klar minoritet. Instruktionerna har antingen en adresseringsmod eller full uppsättning (undantaget sådana som kan ge självmodifierande kod eller icke relokerbar kod, tex flytta till en PC-relativ adress eller bit-testa i ett adressregister). Det finns (för närvarande, det finns 5 lediga kombinationer) 14 adresseringsmoder:

- Data Register Direct
- Address Register Direct
- Address Register Indirect
- Address Register Indirect with Postincrement
- Address Register Indirect with Predecrement
- Address Register Indirect with Displacement
- Address Register Indirect with Index
- Absolute Short
- Absolute Long
- Program Counter with Displacement
- Program Counter with Index Immediate

Här står Index för dataregister+offset byte.

Att beskriva hela Instruktionsuppsättningen blir för långt, så jag tar bara några exempel:


```

MOVEM.L DO-D5/A0-A2,-(SP) ;Spara lite register
MOVE.L #127,DO ;Lägg 127 i DO (enords-instruktion !)
BITEST: BSET #5,(A0)+ ;Testa om biten är noll och sätt den
DBEQ DO,BITEST ;Fortsätt leta efter en nollställd bit
;upp till 128 gånger
PEA OFFSET(A0) ;Spara beräknad adress på stacken
LINK A6,#-10 ;Spara gamla A6, reservera 10 bytes på
;stacken och låt A6 peka på dem
UNLK A6 ;Återställ hela rasket
DIVU 14(A0,D4.L),D3 ;Dividera D3 med innehållet i adress
;14 + A0 + DO (två 32 bitars pekare)
MOVE.B BITEST(PC,D3),DO ;Fiska upp ett värde med PC-relativ
;indexering (D3 betraktat som ord)

```

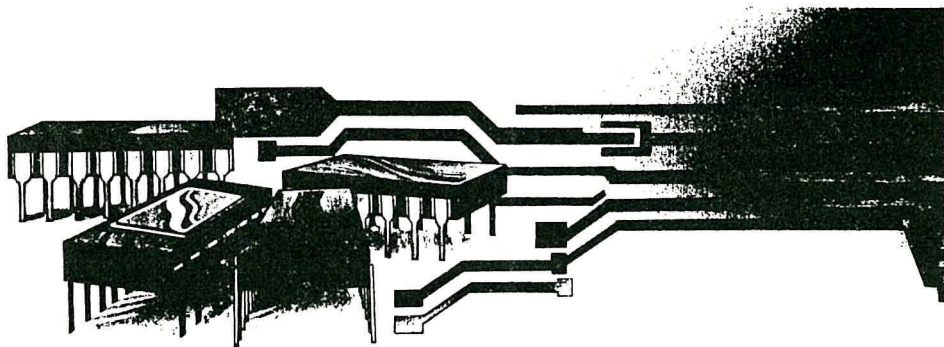
Det finns naturligtvis enklare instruktioner också, fast dom är inte lika roliga att beskriva.

Snart kommer Motorolas MMU som har en intressant (om än inte helt lyckad ur mjukvarusynpunkt) uppbyggnad:

Varje funktionskod (t ex Supervisor Data) från processorn (eller annan bus-master) genererar en intern vektor i MMU-erna (man kan köra flera MMUer nästan transparent). I varje MMU finns sedan 32 st autonoma segment som alla ligger och "lyssnar" på vektorn för att se

om den ligger inom deras område. Om så är fallet kontrolleras med hjälp av en mask om adressen ingår i segmentets logiska adressrymd. I så fall byts de adressbitar i den logiska adressen som man testade ut mot segmentets fysiska adress och adressen är klar. Man kan också få interrupt på accesser i ett godtyckliga segment. Man kan naturligtvis även avläsa om ett segment har använts eller har skrivits in i.

Dan Norstedt



Lispmaskinen CADR

Vad är nu detta? undrar förstås läsaren. En maskin som är baserad på Lisp? Ja, faktiskt: vid AI-labbet vid Massachusetts Institute of Technology har man utvecklat en persondator helt programmerad i Lisp. Som persondator betraktad är den förstås lite mer än vad man vanligen menar med en sådan. Den är avsedd att vara ett forskningsverktyg, och som sådant har den blivit mycket lyckad.

Utseende

Lispmaskinen (som den ser ut i sin ursprungliga konfiguration) består av 3 delar: burk, disk och terminal. Burken består av cpu, unibus-rack och kraftaggregat. Storlek som en garderob ungefär (fast det är mest luft, anledningen till att burken är så stor är mest för att det blir bekvämare att komma åt att peta i den då). Disken kan vara antingen en 80-megabyte-disk eller en 300-mega-dito. Terminalen är uppdelad på skärm och tangentbord. Skärmen är bitmappad, stor som ett A4-papper ungefär (*stor* upplösning). Tangentbordet ser vid första anblicken ut som vilket tangentbord som helst, fast lite stort, men om man tittar närmare på det så ser man många intressanta tangenter, bl.a. *meta*, *hyper* och *super* förutom den vanliga *ctrl*. Den har också särskilda tangenter för *break* (som normalt placerar en i en breakloop), *abort* (som gör att man kommer tillbaka till toploopen), *clear-screen*, etc.

Mikroprogrammerad

56 Lispmaskinen är som alla nya datorer mikroprogrammerad. Denna mikrokod

exekverar maskininstruktioner som inte riktigt ser ut som man är van vid, eller vad sägs om instruktioner som *assq*, *car*, *cdr* & *apply*? Här är ett exempel på vad Lispkod blir kompilerad till. Lispfunktionen som defineras på följande sätt,

```
(defun bar (y)
  (let ((z (car y)))
    (cond ((atom z)
           (setq z (cdr y))
           (foo y))
          (t nil))))
```

kompileras till denna följd av instruktioner (formatet, inklusive kommentarerna, är det som disassemblern ger ifrån sig, någon assembler finns inte!):

```
20 CAR D-PDL ARG|0           ;Y
21 POP LOCAL|0               ;Z
22 MOVE D-IGNORE LOCAL|0    ;Z
23 BR-NOT-ATOM 30
24 CDR D-PDL ARG|0           ;Y
25 POP LOCAL|0               ;Z
26 CALL D-RETURN FEF|6       ;#'FOO
27 MOVE D-LAST ARG|0         ;Y
30 MOVE D-RETURN 'NIL
```

Lisp

Den Lisp som används är en nära släkting till MacLisp. Den har dock många fler features och saknar många av MacLisps misfeatures. Något av det första som märks är det annorlunda formatet på lambda-listor, som kan innehålla lamda-list-"keywords". En funktionsdefinition kan se ut så här:

(defun foo (&rest bar) <body>). Bar binds då till listan av argumenten. Man kan naturligtvis definiera funktioner som motsvarar MacLisps `expr`, och kombinationer är också möjliga. (defun foo (bar &rest baz) <body>) gör att bar binds till första argumentet, och baz till listan av de övriga. Det finns många fler keywords att ha i lambda-listor t.ex. `optional` som talar om att följande argument behöver man inte ge vid anropet. Man kan ange defaultvärde för dessa argument (ett godtycklig uttryck som evalueras). Datatypen `array` finns i den här Lispen, och man har mycket stor kontroll över hur data lagras i dem. Man kan ha arrayer som kan innehålla godtyckliga Lisp-objekt, men man kan också ha arrayer där man specificerar hur många bitar som varje element rymmer (1, 2, 4, 8, eller 16). Strängar finns implementerade som arrayer med 8-bits element. Arrayers storlek kan ändras dynamiskt. Arrayer kan vara indirekta, d.v.s. deras innehåll är definierat som innehållet i en annan array. De båda arrayerna behöver inte ha samma antal dimensioner. Objektorienterad programmering finns i `LispmaskinLisp`. Detta innebär att man programmerar genom att skicka meddelanden till objekt som har "metoder" för att reagera på meddelanden. *Smalltalk*, *ACT1* och *DIRECTOR* är några välkända s.k. actorspråk. I `Lisp-machine` finns "smaker" (eng. *flavors*), för att beskriva objekt. Ett exempel på hur man använder *flavors* (taget ur "Chinual", *Lisp Machine Manual*, av Daniel Weinreb & David Moon):

```
(defflavor ship (x-position
                  y-position
                  x-velocity
                  y-velocity
                  mass)
  ()
  :gettable-instance-variables)
(defmethod (ship :speed) ()
  (sqrt (+ (/char'136 x-velocity 2)
            (/char'136 y-velocity 2))))

(defmethod (ship :direction) ()
  (atan y-velocity x-velocity))
```

Defflavor definierar en ny smak. Dess första argument är namnet på den nya smaken, de övriga är smakens instansvariabler (för den som är bekant med *SIMULA* kan de jämföras med klassattribut). Den tomma listan bryr vi oss inte om här. Resten är ett antal optioner, i detta fall endast en, nämligen `:gettable-instance-options`. Det gör att en metod genereras för varje instansvariabel för att lämna värdet av den instansvariabeln. *Defmethod* adderar en metod till smaken. Den första skapar en metod för att ta hand om `:speed-meddelandet`, den andra en metod för att ta hand om `:direction-meddelandet`. För att skapa ett skepp skriver man t.ex. (`setq my-ship (make-instance 'ship)`). Om man vill veta `x-positionen` för skeppet (efter att man har manipulerat det en stund) skriver man (`funcall my-ship 'x-position`). Man kan även blanda smaker, genom att ta en existerande smak och bygga nya smaker ovanpå denna. Byggandet är dock inte hierarkiskt, fast hur det går till går vi inte in närmare på här.

Worth A Fortune

Editors

Det kanoniska sättet att skriva program på Lispmaskinen är att använda texteditorn, som heter *ZWEI* (Zwei Was Eine Initially, *EINE* = Eine Is Not Emacs), för att mata in sina program, och sedan antingen kompilera eller köra direkt i interpretatorn. *ZWEI* är alltså en texteditor som utgår från *EMACS*, men som förstås är lite annorlunda än *EMACS* (helt enkelt p.g.a. miljön). *ZWEI* är dessutom lättare att kostymisera än *EMACS* eftersom *ZWEI* är gjord i Lisp och inte i *TECO* som *EMACS*. I *ZWEI* finns en del trevliga kommandon som t.ex. "Evaluate Buffer" och "Compile Defun", som man med fördel använder när man ska prova sina nya funktioner. Eftersom ju *ZWEI* ligger i samma Lisp-miljö som allt annat i Lispmaskinen fungerar detta utmärkt.

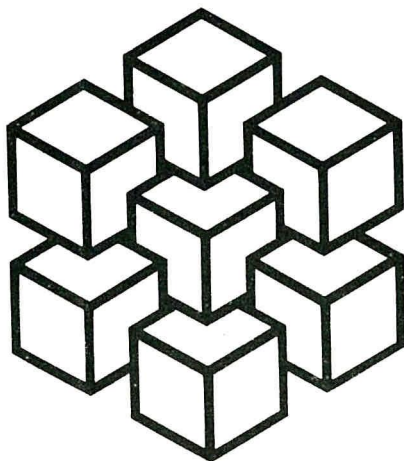
Musen

Till Lispmaskinen finns en liten skojig manick kallad "the mouse". Det är en liten låda ansluten med sladd till tangentbordet. När man rör på lådan, som har en rullboll på undersidan, överförs dessa rörelser till maskinen och gör att en liten pil på skärmen rör på sig. Med hjälp av musen och fönstersystemet kan man sedan göra mycket skojigt. Man för t.ex. upp en systemmenu i ett "pop-up-window" om man trycker på en av musens knappar. I denna menu finns flera alternativ för att t.ex. dela upp skärmen i flera fönster. Man kan med musens hjälp definiera var ett nytt fönster ska hamna och dess storlek.



CHAOSnet

En av ideerna med Lispmaskinen är att varje användare har sin egen processor med minne och att alla användare använder samma filsystem. Genom att kunna komma åt andra användares filer så förlorar man inget jämfört med timesharingssystem, och man vinner fördelen med att det går lika trögt jämt, eftersom man har sin egen processor. Hur genomför man nu detta? Jo man har ett lokalt nät mellan Lispmaskinerna av ethernet-typ. Ett sådant nät består av en koaxialkabel till vilken man ansluter de enheter som ska kommunicera. Det finns alltså ingen maskin som är "master" vilket gör nätet okänsligt för maskinhaverier. Till nätet har man på MIT dessutom anslutit några pdp-10or som fungerar som filsystem åt Lispmaskinerna (förutom sin vanliga uppgift som timesharingdatorer). CHAOSnet finns för operativsystemen ITS, TWENEX, VMS & UNIX. En kuriositet i sammanhanget är att man slipper ställa klockan när man kallstartar en Lispmaskin. Den frågar nämligen alla som finns på nätet vad klockan är!



Köpa en själv?

En sån här trevlig persondator verkar ju lockande, men vad kostar den? Ja, den är ju avsedd som forskningsdator, så priset är på en lite annorlunda nivå jämfört med vad man annars menar med persondator. Mellan 400kSEK och 600kSEK får man räkna med att punga ut, beroende på hur stor disk och hur mycket minne och annan utrustning som man vill ha. Lispmaskinen byggs och säljs av två företag i Cambridge, Massachusetts, USA (utanför Boston i närheten av MIT). Den ena heter Lispmachines Inc. och har funnits längst, det andra heter Symbolics och säljer en något modifierad version vad gäller yttre utseendet, (mindre burk, annan form på terminalen) i övrigt är det samma maskin.

Per Danielsson

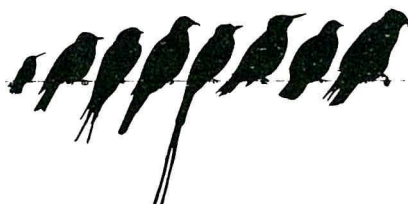


Stackens medlemmar 1982

Ahlen	Björn	Krukmakargatan 42 5tr	117 26 Stockholm	
Ahlqvist	Göran	Rävnäsvägen 42	125 42 Älvsjö	08-477297
Albertsson	Hans	Vanadisvägen 9		
		C/O Fysiska Institutionen	113 46 Stockholm	
Alm	Torsten	Tranholmsvägen 3	170 10 Ekerö	
Backram	Lars	Box 91	661 00 Säfle	
Bengtson	Lars	Box 712	181 07 Lidingö	08-608415
				08-7676396
Bergqvist	Ralf	Snövitvägen 13	161 38 Bromma	08-263776
Boden	Mats	Forskarbacken 15/308	104 05 Stockholm	
Bring	Axel	Neglingevägen 9	133 00 Saltsjöbaden	
Brodda	Claes	Virebergsvägen 5 4tr	171 40 Solna	
Bråding	Lars	Ormingeringen 21	132 00 Saltsjö-Boo	08-7151212
Bystam	Stefan	Drömtigen 17	161 38 Bromma	08-267596
Bäverman	Bengt D M	Västmannagatan 28 2tr	113 25 Stockholm	08-306345
Båge	Mathias	Dalagatan 16	113 24 Stockholm	08-326117
Carlsson	Björn	Rädisvägen 129	162 41 Vällingby	
Croona	Henning	Renfanegränd 21	162 45 Vällingby	08-899409
Danielsson	Björn	Sjövägen 4	184 02 Österskär	0764-60228
Danielsson	Per	Sjövägen 4	184 02 Österskär	0764-60228
Davidson	Hans	Porlabacken 5	124 45 Bandhagen	08-476233
Egnell	Hans	Snärvindvägen 215	162 41 Vällingby	
Ekström	Anders	Ellahagsv 25 A	183 38 Täby	
Ekström	Lars	Lingvägen 222	123 59 Farsta	08-648816
Eldemark	Hans-Erik	Skånegatan 83-85 Bv	116 37 Stockholm	
Eliasson	Thomas	Skogsviksvägen 36	182 35 Danderyd	
Enblom	Sven Erik	Bergslagsvägen 162	161 54 Bromma	
Engqvist	Hans-Lennart	Kraksätrabacken 29	127 31 Skärholmen	
Engström	Dennis	Fredsbergsvägen 3	183 43 Täby	08-838749
Eriksson	Johnny	Bårdgränd 10 Nb	161 47 Bromma	08-878828
Eriksson	Lars-Henrik	Sturegatan 12A 2tr	752 23 Uppsala	018-117303
Euren	Leif	C/O Qvarnström		
		Axvallsvägen 53	121 50 Johanneshov	08-485932
Forsberg	Erik	Skebokvarnsvägen 281 1tr	124 35 Bandhagen	08-861735
Fregelius	Mikael	Ölmevägen 26	123 46 Farsta	
Hagström	Tomas	Gustavsvägen 2	171 49 Solna	08-836048

Hellström	Göran	Dunderbacksvägen 37b	612 00 Finspång	
Hillbo	Anders	Valhallavägen 5/509	114 22 Stockholm	
Honkamäki	Veikko	Artistvägen 4 2tr	122 31 Enskede	08-482104
Insulander	Björn	Vålbergavägen 131	175 63 Järfälla	08-7613341
Jansson	Göran	Jumkilsgatan 14b	752 28 Uppsala	018-101610
Jansson	Mats	Vasavägen 41 1tr	582 33 Linköping	013-132344
Jansson	Torsten	Hammarvägen 62	175 35 Järfälla	
Jensen	Ingvar	Varvsgatan 19	117 29 Stockholm	08-687213
				08-693900
Johnsson	Kjell	Trädgårdsvägen 8	135 53 Tyresö	08-7421225
Julin	Sten-Axel	Floravägen 28	178 00 Ekerö	0756-30036
Kalvik	Ago	Odengatan 72	113 22 Stockholm	08-327738
Koitsalu	Evald	Skansbergsvägen 27	141 70 Huddinge	08-467011
Lagerberg	Peter A	Veckovägen 69	175 41 Järfälla	0758-16031
Landgren	Ulf	Grangärdevägen 4	161 42 Bromma	08-264141
Langlet	Joakim	Styrmansgatan 14a 3tr		
		C/O Bonnier	114 54 Stockholm	
Lansner	Anders	Ljuskärsvägen 105	133 00 Saltsjöbaden	
Levin	Björn	Hodellvägen 17	126 54 Hägersten	08-466072
Liljeryd	Hans	Tomtebogatan 42	113 38 Stockholm	08-325544
Lindberg	Per	Skånegatan 68a 5tr	116 62 Stockholm	08-434258
Lindh	Robert	Terapivägen 16c 9tr	141 56 Huddinge	08-7466031
Lindstrand	Staffan	Champinjonvägen 23	141 46 Huddinge	08-7742524
Ljungdahl	Lars A	Mitisgatan 3	112 37 Stockholm	0755-12624
Lukas	Böök	Trollesundsv 137	124 39 Bandhagen	
Lundh	Bernt	Lilla Sällskapet Väg 159	127 33 Skärholmen	
Lundin	Roberth	Box 138	135 00 Tyresö	08-7122187
Malm	Staffan	Kinnekullevägen 2	161 34 Bromma	08-262249
Mikula	Boguslaw	Handelsvägen 28	122 32 Enskede	
Nilson	Thord	Enevägen 18	153 00 Järna	0755-70657
Nilsson	Hans	Hallongränd 33	182 45 Enebyberg	08-7683805
Nilsson	Martin	Körsbärsvägen 4a/210	114 23 Stockholm	08-154714
Nordström	Hans	Magnekatan 4	195 00 Märsta	
Nordström	Ulf	Klarbärsvägen 3	181 64 Lidingö	08-7661828
Norstedt	Dan	Österåsvägen 4	161 51 Bromma	
Nyman	Bengt	Beckombergaavägen 142	161 53 Bromma	08-875871
Ohlin	Mats	Bastugatan 1	117 20 Stockholm	
Ohlsson	Stefan	Stenkullavägen 54	112 65 Stockholm	08-569432
Persson	Bengt	Maltesholmsvägen 129 2tr	162 37 Vällingby	
Petersson	Lennart	Västgötavägen 5	186 00 Vallentuna	0762-71345

Rend	Dag	Love Almqvists Väg 4a 3tr	112 53 Stockholm	08-503093
Ruden	Tomas	Rosenbjelkenvägen 14	191 77 Sollentuna	08-7542252
Rudholm	Stig	Österledinge 3	762 00 Rimbo	
Rynning	Jan Michael	Klubbacken 36 3tr	128 56 Hägersten	08-468841
Samuelsson	Ulf	Valhallavägen 5/109	114 22 Stockholm	08-153052
Sjöholm	Per	Ekorrv 20	181 44 Lidingö	08-7654675
Sundqvist	Anders	Älsättersgatan 15c:24	582 48 Linköping	
Svanberg	Peter	Adlerbethsgatan 15	112 55 Stockholm	
Svennehed	Hasse	Mariedalsgatan 4a	65220 Karlstad	
Svensson	Erik	Gubbängsv 97	122 45 Enskede	
Svensson	Kurt	Dagsländevägen 17	194 00 Upplands Väsby	0760-81676
Svensson	Lars	Malmvägen 20c 9tr	191 60 Sollentuna	08-066395
Söderman	Nils	Fredriksonsvägen 12	126 54 Hägersten	08-892838
Tirfing	Sören	Rydsvägen 248 A/37	582 48 Linköping	
Viebke	Sven Gunnar	Krögarvägen 16 7tr	145 52 Norsborg	
Virding	Robert	Odensgatan 17	752 22 Uppsala	
Wallberg	Göran	Linnegatan 20	114 47 Stockholm	08-677142
Wallman	Kåre	Parkvägen 12	183 52 Täby	08-7560950
Webbjörn	Mats	Västerlånggatan 57	111 29 Stockholm	08-200502
Wikberg	Valter	Libbyvägen 8	18400 Åkersberga	08-7329383
Wiklund	Örjan	Ingenjörsfirma Bergman & Co A B		
		Box 3318	103 66 Stockholm	
Åberg	Rolf	Skånegatan 78 4tr	116 37 Stockholm	08-444752
Ålander	Ingvar	Kevingeringen 17	182 33 Danderyd	
Årman	Stig	Nockebytorg 25	161 41 Bromma	
Önneflod	Bill	Sätunavägen 14a	195 00 Märsta	



Hackern

Midsommarnattens himmel är ljus
skärmarna scrolla och flimra.
Alla sover i stadens hus
i midsommarnattstimma.
Inga bilar på gatorna går
en ensam klocka i kyrktornet slår,
månen strör silver på taken
endast Hackern är vaken.

Han vaktar troget sitt batchjobb, och ser
hur kompilationen framskrider
alltmedan han operatören ber
om logg på dess länkningstider.
Han sjunger en visa i LisP och Pascal
en såreget sorgsen, syntaktisk trall.
Buggarna frodas syntetiskt
hackern dem patcha frenetiskt.

En ögla av Gödelskt snitt och natur
kan fånga hans nyfikna sinne
med gruppteoretiska fugor ur
ett delbart högsegments minne.
Hans enda övertygelse är
att självreferenser till sanningen bär.
Rekursionen bär ljus i sitt sköte
vid kod och dess datas möte.

Så sitter han vid sin trogna kamrat,
och skriver i Simula, Macro och SAIL,
terminalen vars gröna ljus strålar klart
har för längesen visat hans MAIL.
Ett grafiskt programpaket snyggt plottat har:
modernas topologi är nu klar.
Månen strör silver på taken
endast Hackern är vaken.

(Läsa) nästa inlägg. —

