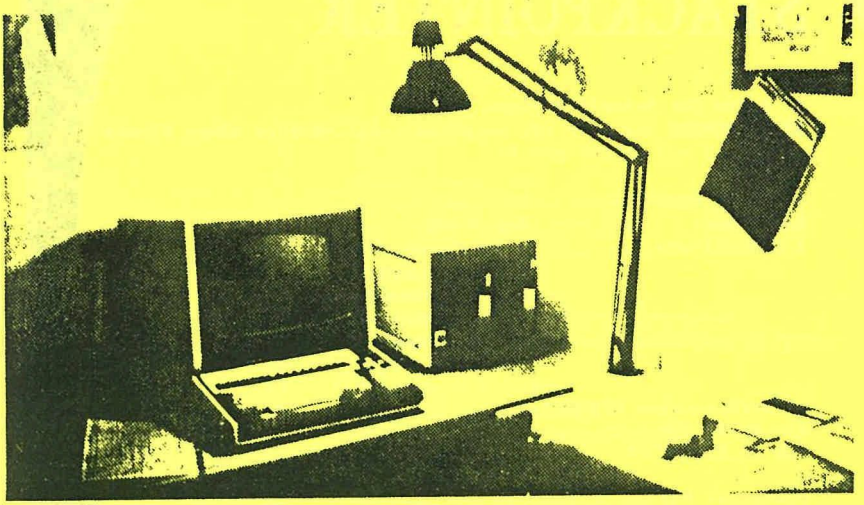
STACKPOINTER

Husorgan för STACKEN, datorföreningen på KTH.
Grundad 1978

2-81



I DETTA NUMMER:

64 sidor!

- ADA-det nya programmeringsspråket
- Skriv ditt eget ADVENTURE!
- Parallellprocesser i LISP
- CP/M på ABC80
- Sagan om AMIS
- Slanglexikon

– Och mycket mer!

MED UTRIVNINGSBILAGA!

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 Danielsson
Per Danielsson
Lars-Henrik Eriksson
Mats Jansson
Per Lindberg

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

Redaktörn har ordet

Ducka! Nu kommer STACKPOINTER, nu tjockare än någonsin!

Ännu ett nummer proppat med intressant computer freak stuff. Vad fattas nu för att STACKPOINTER skall bli en exlusiv kvalitetstidskrift? Kanske en kultursida? Skriv en insändare och tala om vad just DU vill läsa om i STACKPOINTER. Annars kanske det blir en kultursida i alla fall...

För de som väntat med spänning på lösningen till LISP-nötterna i förra numret kommer de här, tillsammans med en artikel om LAMBDA-kalkyl för den som vill lära sig lite om grundvalarna i LISP. Mer LISP: Hur skriver man enklast sitt eget ADVENTURE? Jo, i LISP förståss! Recept i detta nummer. Apropå ADVENTURE och andra CFS-program[1], vad har man för nytta av parallella processer i denna typ av program? Svaret på detta och och hur man skriver ett multi-user ADVENTURE utan att egentligen anstränga sig, finns i detta fullspäckade nummer. Läs också om hur AMIS tog sina första stapplande steg.

Exklusivt! Endast i STACKPOINTER, med ensamrätt i världen! Stackens utsände refererar ett samtal med John McCarthy, fader till LISP, till begreppet Artificiell Intellegens och till timesharingsystemet (Jo, faktiskt!).

Givetvis använder vi textformatterare när vi tillverkar STACKPOINTER. I detta nummer har vi använt programmen SCRIBE, VIDED och VIDEDP. Dessa

---

[1] Computerized Fantasy Simulation

hjälper oss att få rak högermarginal, fetstil, understrykningar m.m. En del rubriker har vi satt med VIDEDP, som plottar dessa med punkten på en DIABLO-skrivare. Extra! Rubriken till denna avdelning är satt med TEX, ett njutoptimerat datorsättningssystem skrivet av datorprofeten D. E. Knuth. Men manuskripten redigerar vi förstås med AMIS.

Som synes handlar de flesta artiklarna i det här numret om programmering och "mjukvara". Det beror antagligen på att kärntruppen i STACKEN finns samlade kring NADJA, en av datorerna på teknis. Mängden av mjukvaruartiklar överenstämmer nog tyvärr inte helt med medlemsskarans intressen.

Därmed inte sagt att vi skall ha mindre mjukvara, snarare mer hårdvaruartiklar. Så, hårdvarukunniga! Hjälp oss med artiklar om prylarna, burkarna, de nya kretsarna, hur man bygger en X med bara Y kretsar etc. Det behöver inte vara något superavancerat för att vi skall ta med det (undvik att skicka doktorsavhandlingar!).

Och, när du otåligt går av och an därhemma i väntan på nästa STACKPOINTER, kom ihåg:

/Red

ORDFÖRANDEN HAR ORDET

Välkommen till detta utökade presentations-nummer av STACKPOINTER! Meningen är att detta nummer skall utgöra en presentation av vår förening och tryckas upp i dubbel upplaga, för spridning utom föreningen. Detta av två orsaker: dels att värva nya medlemmar, dels att ge omvärlden en bild av vad STACKEN är, och hur den fungerar.

STACKEN idag.

När detta skrivs är föreningen 3 år gammal. Trots den korta tiden har massor hunnit hända. Studiebesök, föredrag, samköp, lokal, projekt, och ett unikt utbyte av ideer som aldrig skulle ha blivit av om inte STACKEN fanns. Vi har ett intimt samarbete med vår storasyster datorföreningen LYSATOR i Linköping, och flera andra liknande organisationer. STACKEN idag har nästan 100 medlemmar, alla på sitt sätt verksamma i vad som brukar betecknas som datorrevolutionen. Här finns en anda som kanske påminner om den man kunde finna i radioklubbarna då radion var ung. Eller med andra ord ENTUSIASM.

Vad gör man i STACKEN?

I STACKEN försiggår en massa olika aktiviteter. Det är bara att välja och vraka och plocka ut det man tycker verkar intressant. Här finns studiebesök, föredrag, programmeringsprojekt av allehanda slag, vi har en egen dator att leka med, projektgrupper och en en tidning. För ögonblicket är de största projekten AMIS-projektet, där man arbetar på en avancerad texteditor och 16-bitarsprojektet där man konstruerar en egen 16-bitars mikrodator åt STACKEN och dess medlemmar. Båda dessa projekt beskrivs närmare i artiklar inne i tidningen.

För att ha utbyte av STACKEN måste man förstås inte vara med i någon projektgrupp, man kan programmera på egen hand, träffa likasinnade, och utbyta ideer. Här kan man sätta sig i lokalen och bläddra i en tidskrift. I STACKEN är man en auktoriserad datorentusiast!

Medlemmarna.

Föreningens medlemmar är av den mest skiftande ålder och bakgrund. Tyngdpunkten ligger på universitets- och högskolestuderande, men våra medlemmar återfinns också bland gymnasiestuderande, doktorander och yrkesverksamma inom datorindustrin. Denna bredd ger föreningen stor fackkunskap i datafrågor, både på mjuk- och hårdvaruområdet. Gemensamt för medlemmarna är förstås deras brinnande intresse och kunskap om datateknik (men ingen skall vara rädd att gå med i föreningen för att han är nybörjare. Inflytandet från alla andra medlemmar ökar snabbt på hans kunnande.)

Lokalen.

Vi har äntligen fått en fungerande lokal, visserligen i minsta laget, men ändå. (Elaka tungor kallar den för 6502, mikrodatorn med den minimala stacken...) Den ligger iallafall rätt bra placerad, alldeles intill terminalsalarna. Bland möblemanget märks en egen dator av märket SEVEN-S, ett kylskåp, bokhylla med intressanta böcker och tidskrifter. En bryggare för kaffe och te är på väg. Tyvärr är den för liten för att hålla möten i, så de håller vi i någon av de många övningssalarna som finns på teknis. Det hade dock inte varit så dumt med en större lokal. Vi är inresserade av allt som erbjuds, oavsett storlek och läge!

Varje medlem bör ha en egen nyckel till lokalen. Sådana har nu mångfaldigats, och kan kvitteras ut mot en depositionsavgift på 40 SEK, som återfås då man lämnar tillbaka nyckeln.

Föreningens syfte är Kunskap.

Jag har förut nämnt att våra medlemmar har de mest skilda bakgrunder och intressen. Trots detta finns ett viktigt gemensamt drag: En STACKEN-medlem är en kunskapshungrig individ. Är man intresserad av datorer och programmering, är man alltid nyfiken på nyheter och finesser. Ett slags idemässigt pryldille, kanske? Så ett av föreningens främsta syften är att ge medlemmarna möjlighet att lära sig mer, och att sprida information. Här finns ingen gräns för hur djupt man kan tränga in i datorernas underbara värld. I teorin, åtminstone, finns det inte heller några gränser för vad man kan göra med en dator. (Även om jag är säker på att en teoretisk matematiker inte håller med om det sista...)

Mot högre höjder med ADA

Har ni hört historien från USA om raketen (en mariner-sond) som gick i backen för att någon glömt ett kommatecken i en DO-sats i det FÄRTRAN-program som styrde raketen?

Den historien och många andra fick till slut det amerikanska försvarsministeriet (DoD) att tappa tålamodet. Vid en inventering av de instanser under DoD som sysslade med programmering, upptäckte man att mer än 50 olika programmeringsspåk användes för vad som kallas embedded computers, dvs datorer som sitter inuti olika apparater utan att man egentligen behöver veta om det. Små datorer i dammsugare, bilar, kryssningsrobotar, leksaker m.m. är exempel på embedded computers.

För att få ordning i denna djungel av språk, sökte man därför ett språk som skulle kunna ersätta alla andra språk i framtiden. Något sådant språk hittade man naturligtvis inte, och utlyste därför 1974 en tävling; Hitta på ett språk som har allt det vi behöver!

Jean Ichbiah, en driftig fransman på den tiden anställd vid Honeywell Bull, nappade på utmaningen och presenterade språket GRÖN (neutralt namn). GRÖN som kom att bli det vinnande förslaget, döptes om till ADA efter världens första programmerare Ada Augusta, Lady Lovelace, och presenterades för allmänheten 1979.

ADA är ett mycket stort språk med många ideer tagna från PASCAL. Egentligen innehåller ADA inte mycket som är nytt, utan är mer ett hopkok av konstruktioner och finnesser från tidigare redan existerande språk. ADA följer den s.k. algol-traditionen dvs programmen har blockstruktur, procedurer satser m.m. Vidare finns i språket möjligheter att skapa och hantera parallella processer (flera oberoende samtidigt exekverande programdelar), något som gör det omöjligt att använda ADA under flera av de gamla operativsystem som används nuförtiden. Eftersom ADA bör kunna användas för konstruktion av mycket stora programsystem, har man lagt stor vikt på modularitet, och möjligheten att utveckla och testa olika delar av ett program var för sig, för att slutligen länka ihop dessa till en färdig enhet.

ETT ADA-EXEMPEL

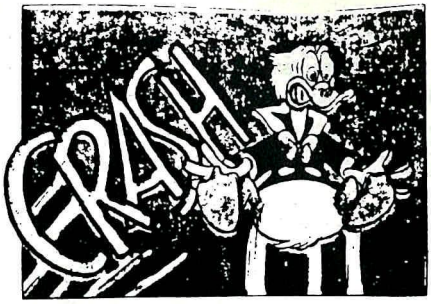
I STACKPOINTER nummer 1-1979 fanns ett programmexempel skrivet i SIMULA och i ALGOL68. Här följer samma exempel skrivet i ADA:

```
-- Ett ADA-program för att göra lite
-- aritmetik på komplexa tal
RESTRICTED(text_io) PROCEDURE complex_math IS
   TYPE complex IS
      RECORD
         re :REAL;
         im :REAL;
      END RECORD;

   ett, i, u, v :complex;
   z :complex := (3,4);

   FUNCTION "+" (u,v :complex) RETURN complex IS
      BEGIN
         RETURN (u.re+v.re, u.im+v.im);
      END "+";

   FUNCTION "*" (u,v :complex) RETURN complex IS
      BEGIN
         RETURN (u.re*v.re-u.im*v.im, u.re*v.im+u.im*v.re);
      END "*";

   BEGIN -- Nu kommer kroppen till complex_math
      ett := (1,0);        -- Initiera några variabler
      i := (0,1);
      u := (z.re, 12);
      v := (z+u)*(ett+i); -- Använd ovanstående funktioner

      PUT("Det komplexa talet v innehåller nu realdel=");
      PUT(v.re); PUT(NEWLINE);
      PUT("och imaginärdel="); PUT(v.im);
      PUT(NEWLINE);
   END complex_math;
```

Detta program ger vid körning utskriften:

```
Det komplexa talet v innehåller nu realdel=-10.00
och imaginärdel=22.00
```

Detta exempel demonstrerar många av ADAs faciliteter. RESTRICTED(text_io) före procedurdeklarationen betyder att man inuti proceduren har tillgång till ett programpaket för in och utmatning. complex_math är den yttersta proceduren, och räknas därför som huvudprogram. Liksom i PASCAL kan man deklarera egna datatyper, i detta fall typen complex som är en enhet bestående av två reella tal re och im (real och imaginärdel). Sedan deklareras variablerna ett, i, u, v och z. För z visas möjligheten att redan vid deklarationen ge variabler ett initialvärde. Utseendet hos värdet till z visar en mycket viktig egenskap hos ADA; Värden i alla datatyper, även strukturerade, kan i ADA skrivas direkt, som literaler. Här är (3,4) ett literalt värde av typen complex.



Ett annat viktigt begrepp i ADA är sk. overloading (överbelastning). Man kan nämligen definiera flera olika versioner av en viss procedure eller funktion, med olika datatyper för parametrarna. Denna procedur eller funktion säges vara overloaded. När en sådan anropas i ett program, så kommer rätt version att väljas beroende på argumentens datatyp. Deklarationen av funktionen "+" visar att även vanliga operatorer kan "överbelastas" på detta sätt. Denna funktion definierar här addition även för operander av typen complex.

Funktionen "*" definierar på samma sätt multiplikation. Efter det BEGIN som kommer efter END "*"; börjar huvudprogrammets kropp. Här börjar exekveringen när proceduren complex_math startas. Vid tilldelningen av v visas hur de definierade funktionerna "+" och "*" används. Vid utmatningen används PUT, en funktion överbelastad med alla fördefinierade datatyper (STRING, REAL, INTEGER etc.).

Nåväl jag har inte hunnit ta upp mer än en bråkdel av ADA här. T.ex. möjligheterna att definiera universalprogrampaket som kan kompileras utan att alla ingående datatyper är bestämda. Dessa bestäms först när paketet skall användas.

Slutligen kan sägas att en av huvudmålsättningarna med ADA var att skapa ett SÄKERT programmeringsspråk. Det råder delade meningar om hur säkert ADA egentligen är. Dels är språket mycket stort, det innehåller många konstruktioner som kan ge upphov till tvetydigheter vid användningen. En i min mening farlig sak är möjligheten att döpa om identifierare med satsen RENAME. Erfarenheter från FÅRTRAN säger att sådana konstruktioner knappast ökar läsbarheten.

Som avslutning kan jag berätta att C.A.R. Hoare, som fick årets Turing award (datalogins nobelpris) i sitt anförande uttalade sig mycket negativt om ADA; ett alltför komplicerat språk för att kunna skriva pålitliga kompilatorer för anser han. han sade sig vara livrädd för de kryssningsrobotar m.m. som skulle styras av ADA-program... (det kanske bör nämnas att Hoare själv ställde upp med språket GUL i ovanstående tävling).

/Dag Rende

John McCarthy besöker Uppsala

I dagarna har Uppsala fått besök av en celeber gäst. Professor John McCarthy från Stanford University hälsade på och höll seminarier inom AI.

Seminariet den 22:a juni om kunskapsmodeller i AI var välfyllt med deltagare som kommit för att lyssna och för att se denne världsberömde AI-forskare. Professor McCarthy ser verkligen ut som man föreställer sig en typisk professor: han är i övre medelåldern, med vitt hår, skägg och glasögon. Det märks att han är en van föreläsare; han är mycket pedagogisk och man har inga svårigheter att höra och förstå vad han säger.

Uppfann LISP och time-sharing systemet

Vad är det då som John McCarthy blivit så berömd för? Kanske först och främst för att det var han som uppfann programspråket LISP i början på 60-talet, medan han ännu befann sig på MIT. Men han är även upphovsman till andra viktiga saker. Få personer vet nog att det var han som uppfann time-sharing systemet i slutet på 50-talet och att det var han som uppfann principen om garbage collection.

Efter seminariet passade jag på att fråga Professor McCarthy om hur det gick till när time-sharing systemet kom till. Ideerna till det hade vi 1958, berättade han. Vi hade en IBM 704 på MIT, och vi lyckades ansluta en skrivmaskin till datorn genom att göra en patch i operativsystemet. Datorn läste in tecken från skrivmaskinen och såg efter om det gick att göra någonting med dem. Det var 1959 och det här första systemet, som egentligen var en föregångare till Time Sharing, kallade vi "Time Stealing". Men, anmärkte McCarthy, mig veterligen hade ingen tidigare gjort ett time-sharing system i den betydelsen att det för varje användare verkar som han hela tiden har maskinen för sig själv. Om man däremot med time-sharing menar att datortiden fördelas mellan olika användare vid konsoler, så fanns det ett sådant system redan 1952, som var ett militärt system avsett för USAs luftförsvar. Men där hade operatörerna bara olika knappar att trycka på, och kunde inte programmera ifrån konsolerna. Professor McCarthy är hela tiden mycket försiktig med att ta åt sig ära för det han har upptäckt, och menar att det kan ju ha varit någon annan som han inte känner till som har upptäckt samma sak.

Först med garbage collection

"Vad var det då som ledde till uppfinningen av garbage collectorn?" frågade jag. Jo, i IPL (ett urgammalt språk som liknade Fortran och som innehöll faciliteter för listhantering) kunde man explicit avallokera minne som använts till listor genom att anropa subrutinenen ERASE. John McCarthy kom på att det ofta gick att skriva stora delar av sitt program som ett enda nästat funktionsanrop (speciellt om man använde en villkorlig funktionsprocedur COND(C,V1,V2) som fick värdet av V1 om C var TRUE och värdet av V2 annars). Det visade sig då vara väldigt praktiskt om man slapp de explicita anropen av ERASE, för då kunde man skriva hela programmet enbart med funktionsanrop, så som man gör när man idag skriver LISP-program.

Man kan inte definiera intelligens idag

Professor McCarthy tystnade ett tag, gungade på stolen och väntade välvilligt på nästa fråga. Man kände sig nästan lite knäsvag så där öga mot öga med mannen som införde termen "Artificiell Intelligens" sommaren 1956, när just nu diskussionens vågor i Sverige går höga om AI. Jag frågade McCarthy om hans uppfattning av AI, i egenskap av att ha infört termen. "Med AI menar jag det som krävs för att datorer skall kunna utföra uppgifter som normalt kräver mänsklig intelligens" svarade han omedelbart. "Men vad är då intelligens? Hur ska man definiera intelligens?" undrade jag. Att definiera intelligens trodde han inte att man kunde göra idag. För det krävs stora forskningsinsatser och nya banbrytande upptäckter, men så småningom borde man kunna definiera det, så att man t ex om ett datorprogram kan avgöra om programmet är intelligent eller inte. Men att försöka definiera intelligens idag är som att definiera Amerika efter första kartan över det.

Maskiner intelligentare än människor?

Kan maskiner bli lika intelligenta som människor? Det finns ju de som har använt Kurt Gödels "omöjlighetsteorem" för att försöka bevisa det: Människan kan alltid fråga datorn om dess axiom (grundläggande räkneregler) och därefter konstruera en uppgift som datorn inte kan lösa. Därför skulle hon vara överlägsen datorn. Men, invände Professor McCarthy, om datorn först frågar människan om hennes axiom kan datorn konstruera en uppgift som människan inte kan lösa. Att försöka bevisa saken på det här sättet är lika löjligt som att säga: "Vi tävlar om vem som säger det största talet. Du börjar!"

Hofstadters bok bra så länge den inte försöker vara seriös

Vi talade också om Douglas Hofstadters bok "Gödel, Escher, Bach - An Eternal Golden Braid". McCarthy tyckte att boken var bra så länge den inte försökte vara seriös. Men han kunde inte förstå varför datorer plötsligt skulle få känslor och neuroser, bara för att programmen blir komplicerade. På frågan om en maskin kan tänkas ha mänskliga känslor, ställde han motfrågan om en termostat blir glad när den lyckas hålla temperaturen i ett rum. Han föreslog att man kunde betrakta det som om det fanns olika ordningstal för känslor: En sten har känslor av ordning 0, alltså inga alls, termostaten har känslor av ordning 1 och en människa har känslor av ordning 2. McCarthy berättade att han faktiskt också skrivit en artikel på detta tema: "Ascribing mental qualities to machines", i Philosophical Perspectives in Artificial Intelligence, publicerad av Humanities Press.

Martin Nilsson

'Twas Burroughs, and the ILLIACS
Did JOSS and SYSGEN in the stack;
All ANSI were the acronyms,
And the Eckert-Mauchly ENIAC.

"Beware the deadly OS, son!
The Megabyte, the JCL!
Beware the gigabit, and shun
The ponderous CODASYL!"

He took his KSR in hand:
Long time the Armonk foe he sought.
So rested he by the Syntax Tree
And APL'd in thought.

And as in on-line thought he stood,
The CODASYL of verbose fame,
Came parsing through the Chomsky wood,
And COBOL'ed as it came!

One, two! One, two! And through and through
The final poll at last drew NAK!
He left it dead, and with its head
He iterated back.

"And hast thou downed old Ma Bell?
Come to my arms, my real-time boy!
Oh Hollerith day! Array! Array!"
He macroed in his joy.

'Twas Burroughs, and the ILLIACS
Did JOSS and SYSGEN in the stack;
All ANSI were the acronyms,
And the Eckert-Mauchly ENIAC.

Sagan om AMIS

eller HUR MAN UTVECKLAR EN EDITOR

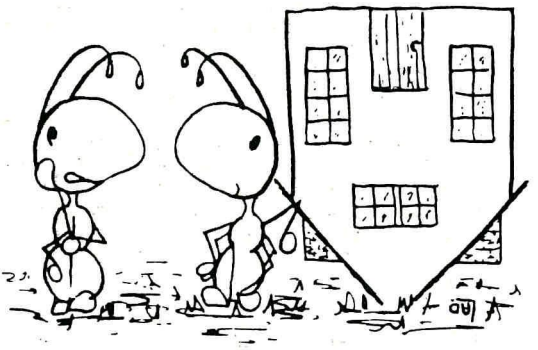
Av **Per Lindberg**

Detta är berättelsen om sju små pojkar som ville göra en editor. Alla här förekommande personnamn är fullständigt avsiktliga, och rejält insyltade. Berättelsen är f.ö. skriven med AMIS själv, så allting här är sant!

Det hela började en gång för länge, länge sedan, i datorernas barndom. Någon hade kommit på att man kunde redigera sina källprogram direkt på datorn (eller elektronhjärnan, som man sa på den tiden). Han skrev sig ett program, med vars hjälp han kunde redigera ett stycke text, vilket som helst, innan han körde det genom kompilatorn, eller vad han nu ville göra med det han skrev. "En strålande ide!", sade alla hans kollegor, och stal den meddetsamma. Alla hade naturligtvis sina egna ideer om hur EDITORn, som programmet kallades, skulle fungera. Så de skrev om och la till, kopierade, plagierade och gjorde om. Och på så sätt kom iden om en editor att göra sitt segertåg världen över. När datorerna blev allt bättre och bättre, kom även editorerna att förbättras, och redan innan BILDSKÄRMSTERMINALERNA gjort den sista SKRIVANDE PAPPERTERMINALEN arbetslös, kom någon på att man borde göra en editor som höll en del av texten konstant uppritad på skärmen, så att man kunde åka omkring i texten och redigera. Den första SKÄRMEDITORN var född. Precis som när den första editorn kom till världen, spred sig iden som en löpeld, och alla försökte göra en egen skärmeditor.

Och sådan var situationen för ungefär ett år sedan, när Lars-Henrik försökte planka den nya fantastiska super-editorn EMACS. Denna EMACS var större, snabbare, bättre, och gladare än någon annan hittills existerande skärmeditor. EMACS kom från datorernas Mecka, USA, och inte nog med det, utan till och med från detta Meckas Kaba, MIT! Till Lars-Henriks förtvivlan fanns inte EMACS tillgänglig på våran dator, NADJA. (Få av oss andra hade vid den tiden hört talas om EMACS, än mindre provat den). Men Lars-Henrik, eller KRSNA som vi kallade honom, var frälst, och ville göra en EMACS som gick att köra på Nadja.

Så småningom fick han ihop något som likande en editor. Skärmuppdateringen var av en ny och revolutionerande typ: Han skrev om HELA SKÄRMEN varje gång något enstaka tecken hade blivit förändrat. Men den fungerade, och vi andra samlades runt Krsnas terminal och skrattade rått.



"This was your first effort at TOP-DOWN design, wasn't it?"

Men nu hade vi blivit nyfikna, och ville lära oss vad det var för något fantastiskt som Krsna hade försökt göra. Så vi samlades tidit en söndagmorgon (Otroligt! Nästan alla vakna!!) och åkte i ett par bilar till Uppsala, där den dator på vilken Krsna hade lärt sig EMACS stod. Den officiella anledningen var dock att vi skulle köra INTERLISP och det nya CFS (Computer Fantasy Simulation)-spelet ZORK. EMACS, INTERLISP, och ZORK kunde nämligen bara köras under operativsystemet TOPS-20, och på Nadja fanns bara TOPS-10. Inom kort var vi alla övertygade om att EMACS nog var rätt bra, och något som vi ville köra även på NADJA. Så gick några veckor, och vi gick och trånade, och längtade tills nästa gång vi skulle åka till Uppsala.

Under tiden hade något annat fantastiskt hänt: ett program som heter KOM hade blivit tillgängligt på Nadja. KOM är ett "telekonferenssystem", det vill säga att i KOM kan man skriva brev till varandra, och det värsta av allt, man kan ha "konferenser" där man gör "inlägg" och "kommentarer". Det hela fungerar ungefär som en insändarspalt, där alla som är "medlem" i en viss konferens kan läsa vad andra skrivit, och själv skriva inlägg. Nu började vi alla med stor entusiasm köra KOM minst 2 gånger om dagen, och producera en astronomisk mängd totalt onyttiga nonsensinlägg..

Men så en dag skrev Krsna ett ilsket inlägg där han förklarade att han var utled på att inte ha en bra skärmeditor på Nadja! Vi skulle skriva en egen EMACS!! Om det bara samlades ett gäng hackers vid Nadja nästa söndag och hackade frenetiskt några timmar, borde vi kunna länka ihop våra moduler någon gång vid midnatt, och en första testversion av vår egen EMACS vara ett faktum! Allt skulle skrivas i Pascal och lite assembler. Samling klockan 12!

En total entusiasm utbröt, och eftersom det var ett par dagar kvar till söndag, ägnade vi våra KOM-inlägg åt att smida planer. Hur skulle vi dela upp projektet mellan oss? Vad ska editorn heta? Skulle den vara en trogen EMACS-kopia, eller skulle vi göra en egen, och kanske ännu bättre editor? Eftersom vi via KOM hade kontakt med ett gäng likasinnade i Linköping, fick vi tillskott av en kille till som var intresserad av att skriva en editor, Anders Ström, av alla kallad A-STROM eftersom han inte använde Ö utan O i sitt user-namn när han körde på TOPS-20 i Linköping. (Han hade allstå också kört EMACS, och insett att allt annat är skräp).

Några förberedelser inför Den Stora Dagen gjordes. Bl.a. fick vi av NADA (Numerisk Analys och DAtalogi, institutionen som har hand om datorn Nadja) ett PPN (konto) på Nadja där vi kunde arbeta och lägga upp våra filer. Det beslöts att projektet skulle utföras i STACKENs regi. (Stacken är datorföreningen på KTH.)

Så blev då klockan 12 denna efterlängtade söndag. Efter n (ett stort heltal) antal akademiska kvartar, var vi sju stycken hackers samlade: Krsna, Per och Björn Danielsson, Erik "Captain Zilog" Forsberg, Johnny Eriksson, A-strom, upprest från Linköping enkom för detta, och jag själv, Per Lindberg (kallad TMPSA eller TMP, efter min ovana att signera mina program "The Mad Programmer Strikes Again"). Och nu bröt en vild diskussion ut! En av de första allvarliga frågorna som raskt avhandlades, var när och var vi skulle äta middag. Den lokala kineskrogen "Minus Ett" sattes som default, tillsammans med ett klockslag. Så småningom enades vi efter mycket vilda (och stundom handgripliga) diskussiner om ett par punkter:

1. Editorn skulle vara en trogen delmängd av EMACS. Inga egna påhitt.
2. Ordförande och projektledare Krsna må, om så krävs, använda livremmen som piska för att få dom andra att hålla klaffen.
3. En skärmuppdateringsmodul skulle anropa en logisk skärmhanterare som i sin tur anropar en fysisk dito.
4. Erik håller klaffen, och slutar prata om ovidkommande saker.
5. Ett subset EMACS-kommandon valdes ut.
6. Övriga moduler utkristalliserades och delades ut.

Efter en stormig middag, satte sig Krnsa vid en terminal, och, ivrigt påhejad av oss andra, skrev en lista på vilka rutiner som skulle finnas i respektive modul.

Nu hade klockan blivit så mycket att A-Strom måste åka med sista tåget tillbaka till Linköping, och vi andra satte oss utmattade framför var sin terminal och läste dagens KOM-inlägg. Klockan blev midnatt, och någon editor hade det inte blivit. Det var nog inte så lätt att göra en editor, iallafall.

Nu hade klockan blivit så mycket att A-Strom måste åka med sista tåget tillbaka till Linköping, och vi andra satte oss utmattade framför var sin terminal och läste dagens KOM-inlägg. Klockan blev midnatt, och någon editor hade det inte blivit. Det var nog inte så lätt att göra en editor, iallafall.

Veckan som följde började vi skriva våra moduler och vilt diskutera vårt projekt i KOM. I KOM löste vi namnfrågan, som efter omröstning (I KOM, såklart!) blev mitt eget förslag: AMIS efter Anti-MISär, ett symboliskt namn på vårt hopp om en bättre framtid. "Editor X" var visserligen ett invant namn, men, trots häftigt motstånd från Björn, ströks det från listan på möjliga namn på editorn. Per Danielsson kontrade dock med en alternativ utläsning av AMIS: "Anti - Mung Interface for Suckers". Krsna ville kalla den TMACS eller TEMACS (ur Tiny EMACS) eller XEDIT. Det senare förslaget visade sig redan vara upptaget av en hemsk editor på CDC-datorerna. Återstod så bara att få honom att ge upp TMACS, vilket lyckades efter en smula elak övertalning. För att ytterligare öka förvirringen kom någon med en ny uttolkning av AMIS: "A Major Innovation In Software".

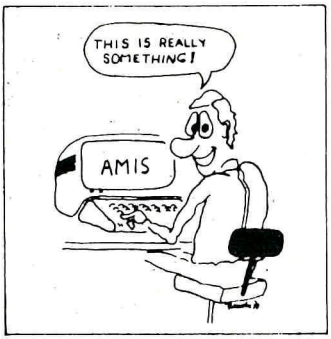
Så blev det söndag igen, och alla samlades för att gå igenom och revidera våra moduler. Inte heller nu blev vi färdiga, men kinesmaten var lika god som förra gången. Söndagarna passerade revy.

Krnsa hade under denna period börjat få för sig att man även kan äta vårrullen med pinnar, och gjorde tafatta försök att skära upp den i små bitar med sina chopsticks. (Han är numera proffs på detta trick!) Ett annat inside-grepp var att man skulle lägga maten i risskålen, och äta med pinnarna i ena handen, och risskålen i andra, nära munnen. Det var under denna period som jag slutade fumla med pinnarna och började använda dom som om det vore den naturligaste sak i världen. (Alternativet hade varit att svälta ihjäl!)

Så en mörk natt när jag satt hemma vid min skärm och körde KOM via telefonlinjen, fick jag syn på ett inlägg av Per Danielsson, som hackade DISKIO, det vill säga den modul som skötte in- och utmatningen på fil. "Break-even point nådd!" Den första modul av AMIS som editerats i sig själv var född! Entusiasmen, som aldrig varit särskilt låg, steg nu, och jag själv började faktiskt tro att AMIS skulle bli något användbart. Det var ungefär nu som åtminstone jag började inse att vad som sagts från några i Linköping (som inte var med i AMIS-projektet) om A-Strom, var sant. Nämligen att han hade 2 st. Lysares kapacitet när det gällde att skriva program. (LYSATOR är datorföreningen i Linköping). Hans modul, den som sköter skärmuppdateringen, visade sig nämligen vara i stånd till de mest vansinniga trix: den skrev ALDRIG om någon del av skärmen där det (av en ren tillfällighet) redan stod vad som skulle vara där.

Andra finesser och featuers diskuterades: bl.a. vad "kill-bufferten" skulle innehålla innan man själv tagit bort någon text? Ett tag innehöll denna buffer (som går att återkalla, därigenom kan man alltså flytta omkring textbitar) innan man tagit bort något innehålla "The Gettysburg Adress", ett känt amerikanskt tal från nationens barndom. Efter en hetsig debatt beslöts att den skulle bort, det skulle bara vara förvirrande för nybörjarna som ville lära sig AMIS. Vi tänkte naturligtvis att alla elever på NADA skulle läras AMIS som första editor. (Denna profetia verkar f.ö. kunna slå in! EMACS åt folket!). Ett nytt talesätt myntades: "-Därför att EMACS gör så!!!!!!" Denna dräpande replik visade sig vara ytterst användbar i alla diskussioner om AMIS, och avslutade snabbt varje dispyt som kunde uppstå. Uttrycket blev så spritt (framförallt i KOM) att det till sist kom att användas i en vidare mening, där varken AMIS eller EMACS behövde vara med. Johnny tillverkade en skylt med de bevingade orden, och satte upp till allmän munterhet.

Projektgruppen hade också fått tillökning i personalen: Jan Michael Rynning, kallad JMR, och Örjan Ekeberg, kallad Örjan. JMR tyckte att vi var lite söliga med att skriva sökrutinerna så han hackade snabbt ihop en egen modul innehållande sökrutiner till AMIS. Örjan kunde inte heller hålla fingrarna i styr, utan hjälpte till med att göra en lathund och hackade också in förkortning av s.k. "Meta-X" - kommandon. Numera är JMR fullt insyltad i projektet, och Örjan är vår femte-kolonnare på NADA.



Nu hade vi en editor som gick att använda, även om mycket återstod att göra. Riktigt HUR mycket hade vi väl inte riktigt klart för oss vid den tiden, annars skulle nog många av oss tappat sugen totalt. Det var nämligen nu som dom VERKLIGA problemen började. Ryktet om AMIS hade spritt sig rejält, och nu ville alla titta på den lustiga lilla apan. Det var alltså dags för en RELEASE!

Den Stora Dagen var som sig bör en söndag, och vi samlades rätt tidigt, vill jag minnas. En ryslig sak slog oss: vi hade ingen MANUAL. Snabbt kasta ihop en sådan, alltså. En total genomgång av samtliga moduler visade att några behövde avlusas lite. Den fruktade "sista buggen" skulle bort innan vi vågade visa upp något. (Den sista buggen definieras i hacker-kretsar som den bugg man tar bort före den sista buggen...) Klockan gick, och vi slet med manual och buggar. Dessutom uppstod vissa organisatoriska problem: vi behövde en speciell verion av ALLTING som skulle sparas som en särskild "release-version", och en speciell area att lägga dessa på. Klockan blev midnatt, och mycket återstod att göra. Så småningom kunde vi ändå vara ense om en relativ bugfrihet, och började sammanställa de färdiga modulerna, och kompilera dessa. I den allmänna röran (vi hade börjat bli lite sömniga allihop) uppstod diverse missförstånd, och klockan gick.

När det började ljusna över taken (det var mitt i vintern då!!) kunde vi till sist meddela Tommy Ericsson som satt som operatör, att han kunde kopiera rubbet. Ytterligare missförstånd: Manualen var skriven med svensk teckenstandard, och alla HELP-filer o.dyl. skulle vara konverterade till "QZ-standard", vilket innebar att vi (D.v.s. de få av oss som var kvar och hållit oss vakna) fick köra dem igenom diverse konverteringsprogram. (Denna konvertering är nu avskaffad på Nadja, och i dagarna måste vi konvertera tillbaka...) Efter ett antal försök lyckades vi, och releasen var ett faktum. Hem och kasta sig på sängen! Naturligtvis var det inte riktigt rätt nu heller, och under måndagen som följde, krävdes ytterligare räddningsaktioner för att ställa saker till rätta.

Och nu började cirkusen ta fart på riktigt! En massa synpunkter och felrapporter började strömma in, och ett speciellt möte i KOM skapades, där synpunkter framfördes och kommenterades. (Ännu i dag är diskussionerna lika intensiva som då, strax efter vår första release.) Det gamla tillhygget "-Därför att EMACS gör så!!!!" användes flitigt. AMIS spreds med ljusets hastighet på det nät av DEC-maskiner som Nadja hänger på, ANF-10. Nu hade dessutom flera fått upp ögonen för att något var i görningen. Grabbarna på ELVIRA, undervisningsdatorn vid E-sekt. på KTH, en PDP-11, var snabbt på plats, och ville få en kopia av källkoden på AMIS för att kompilera med sin egen Pascal. Rapporter om inkompatibiliteter mellan våra olika Pascal:er började hagla. Meningen var ju från början att vi skulle skriva Editor X i Pascal för att 1. få den någorlunda snabb, och 2. för att få den flyttbar till andra system. Nu visade det sig att vi utnyttjat saker i vår Pascal som inte är standard. Mer problem. Och sedan ville Jan Åman och dom andra på Fysikums VAX-11 göra sammaledes. Samma problem igen. Dessa klarades dock upp relativt smärtfritt av de som satt på dessa maskiner och hackade.

Men Sagan om AMIS är inte slut ännu! Fler installationer har frågat efter (och fått) AMIS. Den första internationella ordern kom (via diverse datornät) från Knut Zmaaland på UiO (Universitetet i Oslo:s DEC-10), och ryktet har t.o.m. sipprat ut på det berömda ARPA-nätet i USA. Ett flertal företag som vill skaffa en skärmeditor har frågat efter AMIS. (Iden med skärmeditorer har först efter ett tag spridits till Verkligheten (läs: industrin)). Alla som har en bildskärmsterminal, vill naturligtvis ha AMIS modifierad så att den även går att köra på hans terminalmodell. Och mitt upp i alltihop sitter en liten grupp hackers och skriver förtvivlat på nästa version av AMIS, som ska innehålla ytterligare en delmängd av pappa EMACS. Den färdiga versionen är fortfarande lika avlägsen...

REALITY

REALITY-FANTASY, upside-down design

ADVENTURE I LISP

Tycker du om små gulliga dvärgar? Gillar du att gå vilse i underjordiska labyrinter? Brukar du läsa "Spelunker Today" regelbundet? Tycker du om att göra livsfarliga saker, och sedan återuppstå i ett moln av orange rök? I så fall får du antagligen en rejäl kick av att spela Crowther & Woods' Adventure, stamfadern till alla adventure-program. Adventure gjordes faktiskt ursprungligen i LISP, men översattes sedan till ett mer primitivt språk. Idag finns det massor av efterföljare till detta Adventure, skrivna i alla möjliga språk alltifrån BASIC till Muddle.

Nåt som är lika kul (minst!) som att köra ett Adventure, är att skriva ett eget. Martin NILsson och jag bestämde oss för att hacka ihop ett litet Adventure i LISP. Vi började med att göra en specifikation av vad programmet måste klara:

+ Man ska kunna gå till närliggande rum och platser.
+ Det finns saker som man kan plocka med sej och släppa.
+ Man ska kunna informera sej om omgivningen.
+ Det ska finnas "processer" som agerar självständigt.

För att göra det lätt att implementera bestämde vi oss också att följa några principer:

+ Språk: engelska, för det är lättare att parsa än svenska.
+ Utnyttja MacLisp's finesser så mycket som möjligt.
+ Separera Adventure-scenariot från själva programmet.
+ Programmet ska få plats på en sida. (det blev 5 sidor)

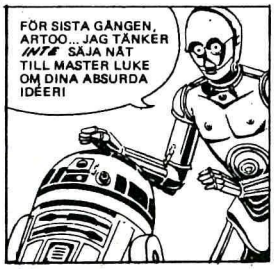
I ett Adventure-program måste man kunna representera prylar på nåt sätt. Det enklaste är att använda en LISP-symbol med prylens namn, och med information om prylen på symbolens egenskapslista. Även såna objekt som personer och platser representeras på det sättet.

På motsvarande sätt är det enklast att låta verb (kommandon) representeras av LISP-funktioner. Då kan man ge kommandon till Adventure helt enkelt genom att anropa funktionen från LISP. Ett exempel:

```
(take banana)
(BANANA TAKEN)
```

Adventures svar är helt enkelt funktionsvärdet som lämnas av funktionen TAKE. Den här metoden har tyvärr en nackdel: Det blir problem med verb som READ och GO, eftersom dom är standardfunktioner som man kanske inte vill definiera om. Dessutom finns det folk som inte tycker om att skriva parenteser omkring all input. Det här löser man genom att införa en funktion som läser en rad och byter alla besvärliga ord mot nånting annat (samtidigt passar man på att byta ut synonymer). Exempel:

```
- take banana
 Banana taken.
```



Inmatningen översätts till (**TAKE BANANA), som evalueras och får värdet (BANANA TAKEN/.) som sen skrivs ut snyggt av en annan specialfunktion.

Hur fungerar nu **TAKE? Något förenklat gör den följande:

```
(COND ((CONTAINSP PLACE OBJ)
       (SUBPROP PLACE OBJ 'CONTENTS)
       (ADDPROP ME OBJ 'CONTENTS)
       (LIST OBJ 'TAKEN/.))
      (T '(THERE IS NO SUCH THING HERE/.)))
```

OBJ är parametern, PLACE och ME är fria variabler. Ok, det där låter ju bra, men antag att man vill ha fler än en banan i sitt scenario? Lätt fixat: bananerna heter BANANA1, BANANA2, o.s.v., och har värdet BANANA på egenskapen INSTANCE-OF. Symbolen BANANA kan då innehålla information som är gemensam för alla bananer.

Ett scenario bygger man genom att sätta upp alla egenskaper som behövs för varje objekt. Det har vi en speciell funktion för: SCENARIO. Exempel på ett scenario:

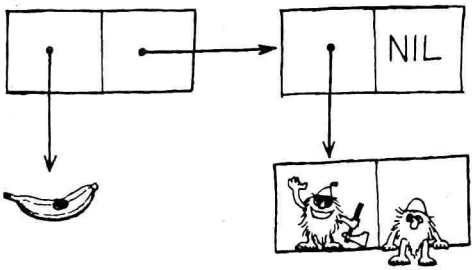
```
(SCENARIO
  (BAR-ROOM DESCRIPTION (YOU ARE IN A LARGE BAR WITH
                         AN EXIT TO THE SOUTH/.)
            EXITS ((SOUTH FOO-ROOM))
            CONTENTS (BANANA))
  (FOO-ROOM DESCRIPTION (YOU ARE IN A LITTLE FOO-ROOM
                         WHICH HAS AN EXIT TO THE NORTH/.)
            EXITS ((NORTH BAR-ROOM)))
  (BANANA PREFIX A
          DESCRIPTION (THE BANANA IS CURVED AND YELLOW/.)))
```

Egenskapen PREFIX talar bara om ifall det heter A BANANA eller AN BANANA. Scenariot har man lämpligtvis i en egen fil. I den filen bygger man också på listan av ord som programmet ska känna till, och definierar eventuellt nya verb.

Om man nu vill att något speciellt ska hända när man släpper bananen, så kan man ange ett lämpligt LISP-uttryck under egenskapen DROP-DEMON (hos bananen):

```
DROP-DEMON (PROGN (SUBPROP ME 'BANANA 'CONTENTS)
                  (ADDPROP PLACE 'ASHES 'CONTENTS)
                  '(THE NITROGLYCERIN-FILLED BANANA EXPLODES
                    WITH A BANG AND LEAVES SOME ASHES ON THE
                    FLOOR/.))
```

På liknande sätt kan man ha demoner till de flesta verb.



Ett kort körningsexempel med ovanstående scenario:

```
- look
 You are in a little foo-room which has
 an exit to the north.

- go north
 You are in a large bar with an exit to
 the south. There is a banana here.

- take banana
 Banana taken.

- drop banana
 The nitroglycerin-filled banana explodes
 with a bang and leaves some ashes on the
 floor.
```

Vi har nu ett fungerande mini-Adventure som uppfyller specifikationen. En del detaljer har jag inte tagit upp, speciellt processer och möjligheten till multi-user-spel, vilket beskrivs i en separat artikel. Hanteringen av texter i beskrivningar och liknande är lite mer avancerad än vad som antyds här. Bl.a. kan man spara minne genom att man istället för (ALLAN TAR KAKAN) skriver öAllan tar kakanö. Det som står mellan vertical bars ("ö" i SIS) utgör en enda symbol i MacLisp.
För närvarande håller vi på att förbättra programmet genom att trimma det lite här och var, och genom att lägga till en parser (fast Martin säjer att det inte är någon parser).

Björn Danielsson

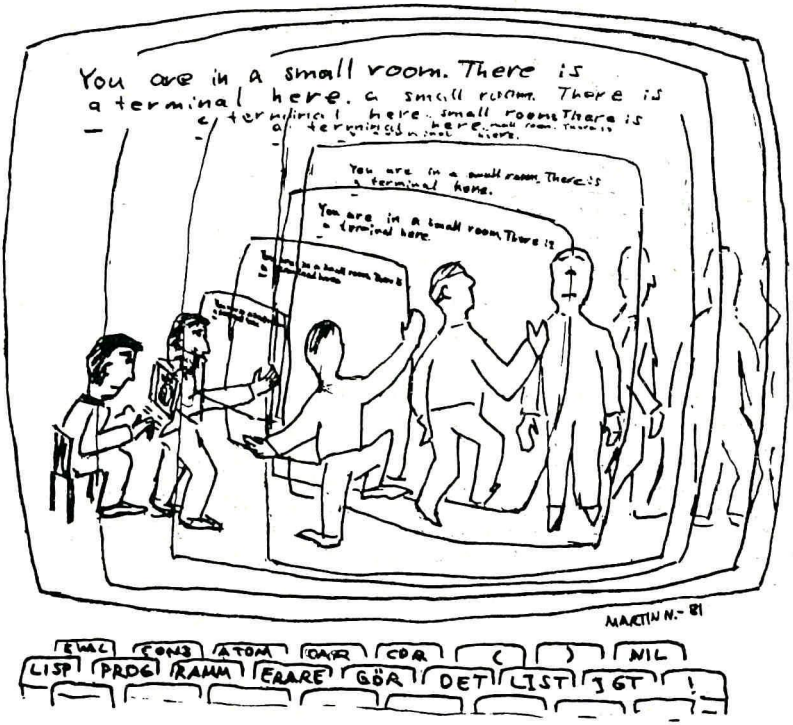
Användning av parallella processer i Computerized Fantasy Simulation program

eller

Hur man gör ett multi-user Adventure utan att egentligen anstränga sig

Många av de som kört programmet Adventure av Crowther och Woods har nog blivit rätt bitna och själva önskat skrivat ett eget sådant. Så även Björn Danielsson och jag. Efter två mediokra försök var, slog vi våra huvuden ihop i slutet på maj. Varför inte ge upp alla storvulna planer, tänkte vi, och istället göra ett LITET enkelt program, och utnyttja alla programspråkets faciliteter? Vi kluddade ner en liten spec, och ...

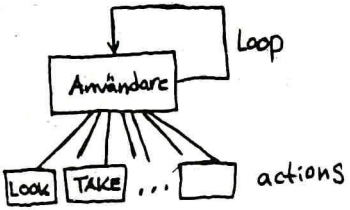
--- POOF! --- <wisp of orange smoke>



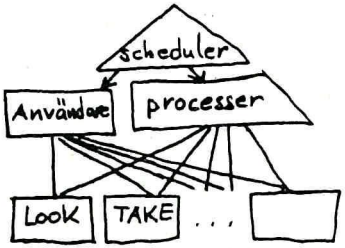
Vad kan man ha för nytta av processer?

Efter några dagar kunde vi köra igång vårt program, som hade alla de nödvändigaste finesserna, och det rymdes på tre sidor (det gäller kärnan av programmet; själva databasen eller scenariot ligger i en separat fil). I vår spec ingick att det skulle finnas en enkel form av (pseudoparallell) processhantering. Tanken var att det kunde vara praktiskt att ha oberoende processer för att implementera t ex levande varelser som dyker upp då och då, ungefär som piraten och dvärgarna i Adventure.

Version 1:
Inga processer
Programmet väntar på input



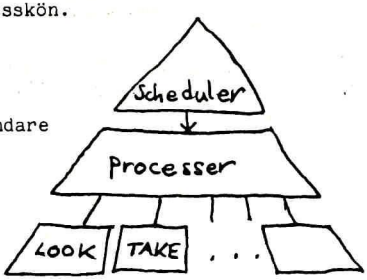
Men den som kör programmet är ju också en levande varelse som borde kunna implementeras som en process. Sagt och gjort. Även användaren blev en process som kunde hanteras på precis samma sätt som andra processer i systemet. Detta faktum gjorde programmet betydligt kortare och enklare.



Version 2:
Processer, en användare
Programmet väntar på input
Scheduler som aktiverar processerna

En annan tanke dök upp: När man nu har användaren som en process, skulle man inte kunna tänka sig att tillåta fler än en användare, genom att skapa flera instanser av processen? Jodå, det visade sig fungera alldeles utmärkt! Till varje användare hör en terminallinje, som öppnas som en fil när processen skapas. För att inte programmet skall stå och vänta på att en speciell användare skall skriva någonting - och därmed tvinga övriga processer att vänta - tittar programmet först efter om det finns någonting att hämta från den linjen, annars fortsätter det med att aktivera nästa process i processkön.

Version 3:
Processerna inkluderar flera användare
Ingen väntan på input
Dynamisk prioritering av processer (möjlighet att prioritera ner processer som kört mycket)

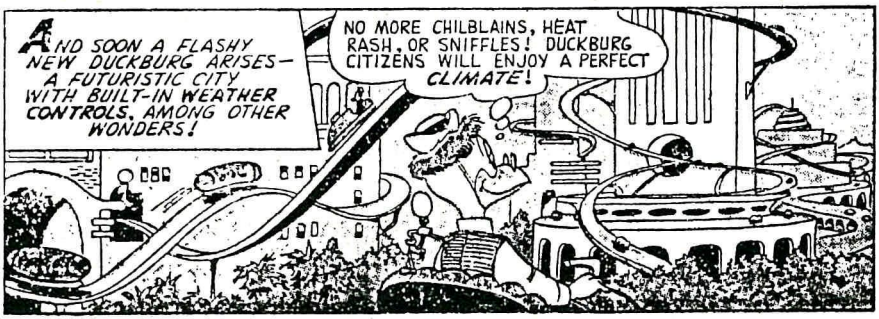


Exempel på hur processer används i ADVENTURE

Processer, eller snarare prototyper för processer, definierar man med hjälp av funktionen DEFPROCESS. Så här definierar man i princip en deltagare i spelet genom processtypen ADVENTURER:

```
(DEFPROCESS ADVENTURER
            (PLACE TYI TYO)
            (PRINPROMPT (**LOOK))
            L1
            (COND ((= 1. (LISTEN TYI))
                   (PRINPROMPT (OR (EVAL (CANONIZE (ADVREAD)))
                                   '(I DON/'T UNDERSTAND/.)))))
            (PASSIVATE 'L1))
```

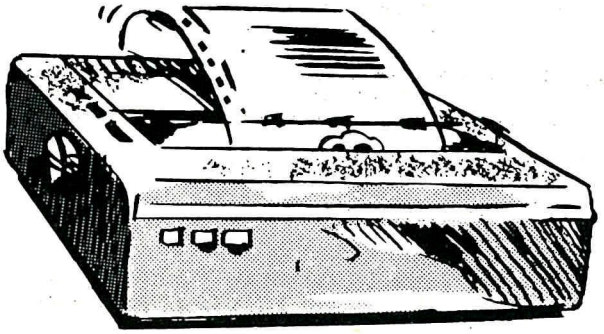
PLACE, TYI och TYO är attribut, där PLACE är platsen som personen befinner sig på, TYI är infilkanalen och TYO är utfilkanalen. Det första anropet av PRINPROMPT skriver ut hur det ser ut där man befinner sig när man aktiveras första gången. LISTEN är en funktion som returnerar 1 om det finns någonting att läsa på infilkanalen, och 0 annars. Om det finns något att läsa läses ett kommando med ADVREAD, det utförs och resultatet skrivs ut. Funktionen PASSIVATE ser till att processen stoppas tillbaka in i processkön och överlämnar kontrollen till schedulern. Parametern till PASSIVATE talar om var exekveringen ska fortsätta när processen reaktiveras.



Instanser av processer kan skapas med funktionen CREATE. Anta att personerna ACKE och LOTTA vill köra ADVENTURE. Om det är Lotta som kör ADVENTURE-programmet och Acke sitter vid TTY17: skapas två ADVENTURER-process-instanser med

```
(CREATE 'ADVENTURER 'LOTTA 'FACTORY TYI TYO)
(CREATE 'ADVENTURER 'ACKE 'KITCHEN (OPEN 'TTY17: 'IN)
                                   (OPEN 'TTY17: 'OUT))
```

TYI och TYO är globala systemvariabler i MacLisp och har filkanalerna till terminalen som värden.



Hur processerna har implementerats

Funktionen DEFPROCESS definierar en ny funktion som blir själva processkroppen. Processens attribut blir parametrar till funktionen. Attributens namn läggs även på egenskapslistan.

CREATE definierar inte en ny funktion, utan lägger endast in namnet på processtypen på instansens egenskapslista. Där läggs även de aktuella värdena på instansens attribut. CREATE stoppar in instansen i processkön.

När en instans aktiveras tas den ut ur kön och prototypens funktion anropas med attributen som argument.

PASSIVATE uppdaterar attributens värden och stoppar tillbaka processen i kön. Även fortsättnings-labeln sparas på egenskapslistan.

Schedulern är helt enkelt en loop som startar processerna i kön i turordning.

Att attributen är explicit tillgängliga för andra processer är praktiskt: Om Lotta vill säga någonting till Acke, kan hon skriva till exempel

- SAY Vad blir det för mat idag?

Om Acke befinner sig i samma rum, får han utskrivet:

Lotta says: Vad blir det för mat idag?

Funktionerna som hanterar SAY gör så att de letar reda på de andra processer som kan ta emot ett meddelande (ADVENTURERs som befinner sig i samma rum), plockar fram deras utkanaler och skriver ut meddelandet på dem.

Framtiden för CFS-spel

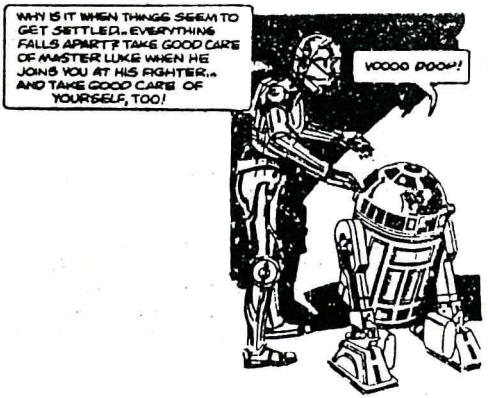
Vår implementation av ett CFS-spel för flera användare är ganska enkel, men fungerar utmärkt. Ännu är dock långt ifrån alla problem lösta, och det finns mycket kvar att göra. För att citera ett stycke i artikeln "Zork: A Computerized Fantasy Simulation Game" av P. David Lebling, Marc S. Blank och Timothy A. Andersson i Computer april 79:

"A still more ambitious direction for future CFS games is that of multiple-player games. The simplest possible such game introduces major problems, even ignoring the mechanism used to accomplish communication or sharing. For example, there are impressive problems related to the various aspects of simultaneity and synchronization. How do players communicate with each other? How do they coordinate actions, such as attacking some enemy in concert?

Putting aside implementation problems, a multiple-player game would need to have (we believe) fundamentally different types of problems to be interesting. If the game were cooperative (as are most D&D scenarios), then problems requiring several players' aid in solving them would need to be devised. If the game were competitive and like the current Zork, the first player to acquire the (only) correct tool for a job would have an enormous advantage, to give just one example. Other issues are raised by the statistic that the average player takes weeks and many distinct sessions to finish the game; what happens to him during the time he is not playing and others are?

We believe there is a great future for this type of game, both for the players and for the implementers and designers of more complex, more sophisticated, and - in short - more real simulation games."

Martin Nilsson



Det är nu möjligt att köra

CP/M på ABC80

Med en tillbyggnadssats som monteras i tangentbordet och ett 80-kolumnerskort som kan monteras floppy-diskenheten får man en RAM-maskin med 48 K minne. För att komma in i CP/M kör man ett BASIC-program "BOOT" och datorn hanteras nu av ett annat operativsystem.

För de som kört större datorer, t. ex. på QZ eller liknande system, känner lätt igen sig, då CP/M i mycket liknar operativsystem på stordatorer. Man kan ochså gå den andra vägen, och säga att det är bra att bekanta sig med CP/M för dem som vill förbereda sig för arbete med större datorer.

CP/M på ABC80 är tänkt som ett starkt komplement till dem som redan har investerat i en ABC80. Det är ingen hemlighet att CP/M är det i särklass mest använda operativsystem för mikrodatorer. Det är också det operativsystem som har det största utbudet av proggramvaror för mikrodatorer.

Några exempel på programvara som arbetar under CP/M:

BASIC-80	interpreterande BASIC
BASIC-80	kompilator till BASIC-80
FORTRAN-80	ger maskinkod
COBOL-80	ger maskinkod
PL/I -80	ger maskinkod
PASCAL/MT	ger maskinkod
PASCAL-Z	ger maskinkod
WORD-STAR	ordbehandlingsystem förmodligen den starkaste på hela marknaden.

För den som vill veta mer om CP/M i allmänhet rekommenderas läsning av MIKRODATORN Nr: 2, 3, 4, 5. Juni-nummret av BYTE. Nästa nr av ABC-bladet där jag kommer att beskriva systemet mera i detalj. För den som vill lära sig CP/M rekommenderar jag läsning av min "HANDBOK för CP/M" på svenska.

För den som vill köpa ett system, eller för den som vill ha mera information om CP/M på ABC80 kan ringa mig. Systemet säljes inte av LUXOR eller METRIC.

För dem som är intresserade av att prova hur det känns att köra CP/M, kan BATCH-köra mot mig över telefon. Jag behöver veta när du vill köra. Det uppstår i och för sig samma problem som vid körning mot annan dator som förutsätter att du har 80 kolumner. Det går att komma runt med att skrolla skärmen i sidled med T80PRT.

Har man ett eget system har man samtidigt en 80-kolumners skärm som används både vid körning under CP/M och ABC80. Det är viktigt att påpeka att det fort farande är en ABC80, allt som du kunde göra förut kan du göra nu. Skillnaden är den att du har uppgraderat ditt system. När du kör systemet som vanlig ABC80 har du tillgång till 32 K minne, printerrutin i prom samt tillgång till extra kommandon:

```
AUTO       automatisk radnumrering
BOFA       BOFA-pekarens adress
BOFA  adr  sätter BOFA till adr
CAT        catalog
CONT       CONTINUE   efter <ctrl C>
CROSS rX   listar rader med ref till rX
CROSS vX   listar rader där vX ingår
DEL        tar bort valfria radnummer
ED         skärm-editor, räddar felakt-
           tiga BASIC-rader.
EOFA       EOFA-pekarens adress
EOFA  adr  sätter EOFA till adr
FREE       ledigt RAM mellan EOFA-STACK
LEN        längden av programmet
LEN   rX   längden av raden rX
RUN   rX   startar exekvering på rad rX
VAR        listar alla variabler
<ctrl L>   rensar skärmen.
```

Veikko Honkamäki

tel: 08 - 48 21 04

DATATEKNIK I TIDEN

FORFARM

För alla er som tycker att FORTRAN är det enda hederliga programmeringsspråket kommer här en välkommen nyhet, FORFARM, lämpat för allt utom datalogiskt strunt. Med extra tilläggsregler, med nya fantastiska dumpmöjligheter kort sagt allt som en gammal god programmerare kan önska sig.

Här följer ett axplock ur FORFARMs faciliteter:

Inaktuella parametrar gamla avlagda parametrar som mist sitt värde tilldelas automatiskt vanligen värdet 7.0 . Formella parametrar tilltalas med "NI" och skall alltid (regel 5.3.4) skrivas med stor bokstav först.

I FORFARM gäller att om man, då man ska börja sin FORMAT-sats med 1X, istället skriver 1-X, så får man på varje ny sida i utmatningen ut texten "FORFARM". Om man istället skriver 1+X så får man ut filen i form av ettor och nollor.

I FORFARM slipper man dessutom deklarera sina variabler. Om det är nåt särskilt man vill säga om dem, så skriver man det efter variablerna inom $CC ... CC$.

Kommentarer är helt slopade i FORFARM. Eftersom detta är ett så klart språk, så är kommentarer överflödiga.

Om du vill gå ut och röka, kan du använda kommandot PAUSE. Då slocknar alla terminaler kopplade till systemet.

Med NONEPRECISION kan man häva en tidigare deklarerad DOUBLEPRECISION-variabel. Om variabeln som man NONEPRECISION-deklarerar tidigare inte var DOUBLEPRECISION, avrundas variabeln uppåt till närmaste oktala tal.

I alla FORFARM-system fins en standardsubrutin som heter TELEPHONE. Om man anropar den genom CALL TELEPHONE och som parameter lämnar ett nummer, ringer det i den telefon som har detta nummer. Om man istället anger nåt annat som parameter, så kan lite av varje hända.

/Classe Brodda

Handsoffen C

Das Komputer is nich
und Mittengrabben. Is
der Springenwork, Blo
Loppencorken mit Spi
nicht Geworken by da
ist yust Rubbernecken
Keepen Hands in das
und Watchen das Blin

Dummkoffen !

t for Gefingerpoken

t easy to Schnappen

wenfusen und

ttzensparken. Ist

s Dummkoffen. If das

und Sightseeren:

Pockets, Relaxen,

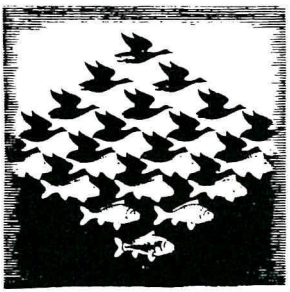
kenlights.

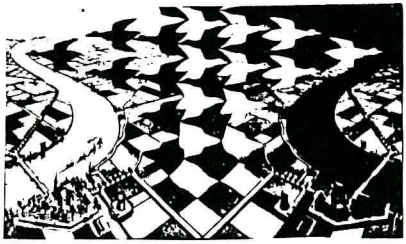
Här fortsätter STACKPOINTERs teoriserie med:

LISP OCH LAMBDAKALKYL

Som väl de flesta läsare av STACKPOINTER har förstått av de många LISP-artiklar som förekommit, så är LISP ett intressant språk ur flera synpunkter. LISP är intressant dels för att program och data representeras på samma sätt, vilket gör det lätt att manipulera och skapa nya program, dels för att det är lätt att utvidga och modifiera, men också för att det är utformat efter en matematisk formalism, kallad "LAMBDA-kalkylen".

Lambda-kalkylen skapades av den matematiska logikern Alonzo Church och beskrevs år 1941 i en skrift med titeln "The calculi of lambda-conversion". Lambda-kalkylen var avsedd att ge en metod att matematiskt beskriva beräkningar, på samma sätt som Turingmaskiner och några andra formalismer[1] . Church's bok är både svår att få tag i och svårläst. John McCarthy, som uppfann LISP, har själv förklarat att han inte begrep hälften av vad som stod i den (men tydligen så begrep han tillräckligt). Den som skulle bli intresserad nog att vilja läsa boken på egen hand kan antagligen hitta den i något universitetsbibliotek.





Vad har då man för nytta av att kunna något om lambda-kalkylen? Det närmast till hands liggande skälet är väl att man skall få en djupare förståelse för hur LISP fungerar och varför det ser ut som det gör. Men kalkylen är förstås inte bara till glädje för LISP-fantaster. Även "vanliga" datorintresserade kan ha intresse av att känna till den. Lambda-kalkyl används nämligen inom flera områden av datalogin och man kan även hitta den inom AI. Man skulle t.o.m. kunna säga att det hör till en datalogs allmänbildning att känna till det grundläggande om lambda-kalkylen lika väl som predikatlogiken.

---

[1] Se min artikel i förra numret av STACKPOINTER.

Centrala begrepp i lambda-kalkylen, liksom i LISP, är funktioner och funktionsapplikation (motsvarar funktionsanrop). Funktioner definierar man med den s.k."lambda-operatorn" (λ), vilken gett kalkylen dess namn. Lambda-operatorn känner man igen från atomen LAMBDA i LISPs lambdauttryck.

Lambda-kalkylen kan alltså något förenklat beskrivas som ett slags "programmeringsspråk" där alla beräkningar sker med funktionsanrop. "Exekveringen" (eller reduktionen) av ett lambda-uttryck sker genom att hela uttrycket byts ut mot den "anropade" funktionens funktionskropp, med alla argumentvariabler utbytta mot de argument som gavs i funktionsanropet, se exemplet nedan!

För att slippa införa en ny syntax och för att göra den följande beskrivningen lite mer lättläst, kommer jag att använda LISP-syntax i stället för Church's syntax, vilken är den som vanligen används för uttryck i lambda-kalkylen. Läsaren bör alltså tänka på att även om uttrycken är väldigt LISP-lika, så är de inte LISP.

En funktion i lambda-kalkylen definieras på likande sätt som i LISP, exempelvis:

```
(LAMBDA (X) ..X..)
```

Detta uttryck är en funktion av ett argument, X, och med funktionskroppen "..X.." (här anger prickarna att funktionskroppen består av mera än X, men att vi inte är intresserade av det andra för ögonblicket). En funktion i den rena lambda-kalkylen har alltså inget namn, utan hela funktionen måste skrivas ut varje gång den skall användas (den kan dock bindas till en lambda-variabel, se nedan).

Precis som man i LISP anropar en funktion, kan man här "applicera den på sina argument", som den exakta termen lyder[2], genom att skriva på följande sätt:

```
((LAMBDA (X) ..X..) foo)
```

Här har vi applicerat funktionen på argumentet "foo". Detta innebär att hela uttrycket byts ut mot procedurkroppen "..X.." där X överallt har bytts ut, eller substituerats, mot argumentet "foo". Vi får alltså hela uttrycket utbytt mot "..foo..". Hade argumentet varit "bar" istället, hade funktionskroppen förstås istället blivit "..bar..". Man säger att det ursprungliga uttrycket reducerats. I LISP skulle det vara för tidsödande att verkligen byta ut X överallt, istället har man låtit X bli en variabel med värdet "foo", man har lambda-bundit variabeln X till "foo", så att varje gång X förekommer i LISP-programmet så används "foo" istället.

---

[2]Här hittar den LISP-intresserade ursprunget till namnet på LISP-funktionen APPLY

I exemplet ovan så angav jag inte närmare hur funktionskroppen såg ut, jag använde också "foo" som argument till funktionen. I lambda-kalkylen, så skulle båda dessa vara andra lambda-uttryck, eller en lambda-variabel. Den enda "datatyp" som finns är nämligen lambda-uttryck! Detta kanske verkar som en svår begränsning, men tänk på hur långt en dator kommer med bara ettor och nollor. Tricket är att låta olika lambdauttryck representera data, precis som man låter olika kombinationer av ettor och nollor representera, tal, teckensträngar m.m.

Om man istället för att "applicera ett lambdauttryck på argument" ger det som argument till ett annat lambdauttryck, så kommer det förra lambdauttrycket att ersätta lambdavariabeln överallt i det senare uttryckets funktionskropp.

Lambda-uttrycken kan alltså vara både aktiva (funktioner) eller passiva (argument), beroende på hur de används. Ett fullständigt exempel bör klargöra sammanhangen:

Antag att man har följande uttryck:

((LAMBDA (X) (X X)) (LAMBDA (Y) Y))

Genom att utföra en reduktion, så ersätts hela uttrycket av funktionens (LAMBDA (X) (X X)) funktionskropp, med lambdavariabeln X utbytt mot argumentet (LAMBDA (Y) Y). Vi får alltså:

((LAMBDA (Y) Y) (LAMBDA (Y) Y))

Om du inte förstår hur det här gick till, så tänk på att vi fick kvar procedurkroppen (X X), där X skulle bytas mot (LAMBDA (Y) Y).

Nu kan vi utföra ytterligare en reduktion! Vi får

(LAMBDA (Y) Y)

och nu kan vi inte utföra några ytterligare reduktioner. Resultatet av att reducera det ursprungliga uttrycket blev alltså (LAMBDA (Y) Y).

Här kan vi stanna upp och dra en paralell till LISP: Våra uttryck har haft formen (LAMBDA (arguments) body) resp. (function arguments..), detta är precis samma som i LISP. Skillnaden ligger i att vi använt ensamma lambda-uttryck som argument, något vi inte kan göra i LISP. I LISP får vi istället skriva (FUNCTION (LAMBDA ...)), för att hindra att LISP tolkar LAMBDA som ett funktionsnamn. I lambda-kalkylen finns överhuvudtaget inga funktionsnamn så inga missförstånd kan uppstå och speciell markering (typ FUNCTION) är onödig.

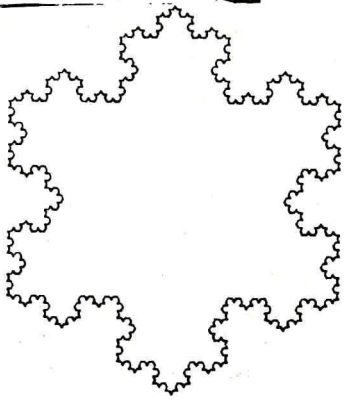
För att återgå till lambda-kalkylen kan vi fundera på i vilken ordning olika reduktioner skall utföras. I exemplet hade vi aldrig mera än en reduktion att välja på vid varje steg, så något problem uppstod aldrig. I allmänhet har man dock flera alternativ, som i uttrycket

```
((LAMBDA (X Y) X)
   (LAMBDA (A) A)
   ((LAMBDA (F) (F F)) (LAMBDA (F) (F F))))
```
[3]

Notera att om det andra argumentet till det yttersta lambdauttrycket, ((LAMBDA (F) (F F)) (LAMBDA (F) (F F))), reduceras först, så får man tillbaka samma uttryck igen, och ytterligare samma reduktion gör att vi loopar. Om man istället reducerar den ytters ta funktionen först, får man:

```
(LAMBDA (A) A)
```

Man kan visa att alla olika ordningar av reduktioner som inte loopar ger samma slutresultat. Det finns också en speciell ordning att utföra reduktioner på, s.k. normalordning, som garanterat alltid lyckas reducera uttrycket utan att loopa, om det överhuvudtaget är möjligt.



[3]Här använder jag två argument. Egentligen får man bara ha ett argument i den ursprungliga lambda-kalkylen, men det eftersom man kan göra om uttrycket till ett krångligare sådant med bara en-argumentsfunktioner, så brukar man tillåta sig att använda flera argument.

Nu när vi har grundläggande kunskaper i vad lambda-kalkylen är, skall vi försöka använda den till något. Som jag förklarade i början av artikeln, uppfanns lambda-kalkylen för att beskriva beräkningar. Detta innebär att alla tänkbara beräkningar kan beskrivas med hjälp av lambda-kalkyl. Påståendet låter kanske lite otroligt, eftersom lambda-kalkylen saknar villkorliga uttryck, datastrukturer och annat som man är van vid, men det är faktiskt ändå sant. För att visa detta skall jag visa hur man kan göra villkor i lambda-kalkylen och hur man kan bilda en enkel datastruktur.

Först villkoret. Dels måste vi ha själva villkorsuttrycket, dels de logiska värdena "sant" och "falskt" som vi skall testa på. Jag påstår att ett villkorsuttryck motsvarande if p then a else b, ser ut på det här sättet:

```
(p a b)
```

Ser detta allt för enkelt ut? Nu kommer "sant" och "falskt":

```
(LAMBDA (X Y) X)    Sant
(LAMBDA (X Y) Y)    Falskt
```

Om "p" är "sant" så skall alltså villkorsuttrycket reducera till "a". Vi provar!

```
((LAMBDA (X Y) X) a b)
```

Här binds X till "a" och Y till "b". Eftersom funktionskroppen endast består av X, så blir resultatet "a", precis som vi ville! På liknande sätt visar man att (LAMBDA (X Y) Y) verkligen betyder "falskt".

Iden här är alltså att det logiska värdet i själva verket är en funktion som tar de två alternativen som argument och som reduceras till det rätta alternativet.

Det andra exemplet var att skriva en enkel datastruktur. Som exempel skall jag ta LISPs CONS-cell. Denna är i princip en record med två element som kallas CAR och CDR. Vi definierar funktioner för att skapa en CONS-cell (CONS) och för att ta fram CAR- resp. CDR-delen (CAR, CDR). Funktionerna ser ut så här:

```
(LAMBDA (X Y) (LAMBDA (P) (P X Y)))     CONS
(LAMBDA (C) (C (LAMBDA (X Y) X)))       CAR
(LAMBDA (C) (C (LAMBDA (X Y) Y)))       CDR
```

Läsaren får själv lura ut hur dessa fungerar. Jag skall bara med ett enkelt exempel bilda CONS av "a" och "b" och sedan visa att CAR-delen av resultatet blivit "a".

```
((LAMBDA (X Y) (LAMBDA (P) (P X Y))) a b)     CONS av a och b.
(LAMBDA (P) (P a b))                          Efter reduktion.
```

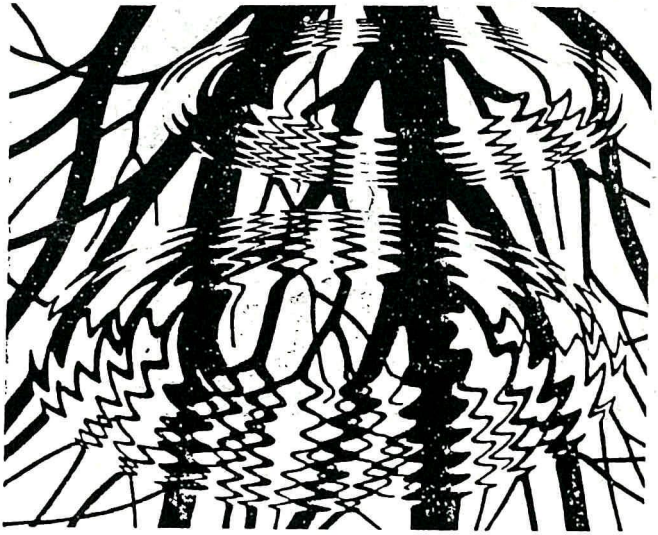
Det sista uttrycket är alltså CONS av "a" och "b". För att bevisa detta tar vi CAR av uttrycket: (varje rad visar uttrycket efter succesiva reduktioner)

```
((LAMBDA (C) (C (LAMBDA (X Y) X))) (LAMBDA (P) (P a b)))
((LAMBDA (P) (P a b)) (LAMBDA (X Y) X))
((LAMBDA (X Y) X) a b)
a
```

Vilket skulle bevisas.

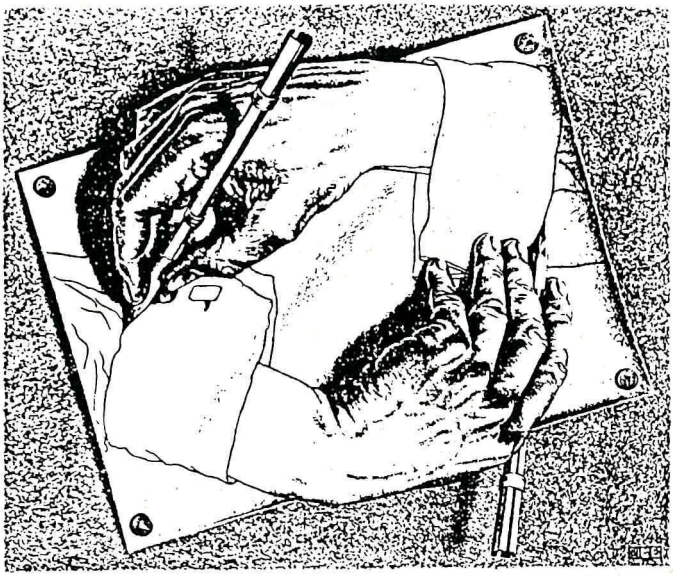
Tyvärr känner jag inte till några referenser som behandlar lambda-kalkylen speciellt utförligt, utom Church's bok. Däremot så finns det många böcker där lambda-kalkylen beskrivs. Om någon skulle vilja läsa mera, så ta kontakt med mig så kan jag kanske föreslå någon lämplig bok för fortsatt läsning.

/Lars-Henrik Eriksson



SVAR PÅ STACKENS LISPNÖTTER

I förra numret av STACKPOINTER gav jag tre olika LisP-nötter som du kunde försöka lösa i Pure LisP. Nu har det blivit dags att presentera lösningarna. Där inget annat anges kan lösningarna direkt köras i MacLisP.



1. Problem: Skriv ett LisP-uttryck som evaluerar till sig självt.
   Lösning, presenterad av Dag Rende:

```
((LAMBDA (X)
         (CONS X
               (CONS (CONS (QUOTE QUOTE)
                           (CONS X NIL))
                     NIL)))
 (QUOTE
  (LAMBDA (X)
          (CONS X
                (CONS (CONS (QUOTE QUOTE)
                            (CONS X NIL))
                      NIL)))))
```

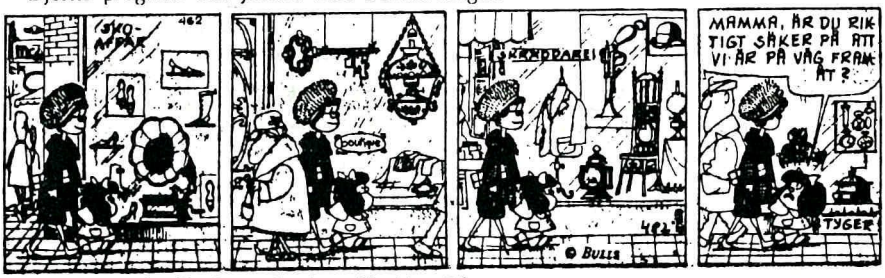
2. Problem: Skriv en funktion som gör samma sak som funktionen REVERSE, utan att använda några LAMBDA-uttryck inne i funktionen och utan hjälpfunktioner.

Lösning, presenterad av Björn Danielsson:

```
(DEFUN REVERSE (X)
  (COND ((NULL X) NIL)
        ((NULL (CDR X)) X)
        (T
         (CONS (CAR (REVERSE (CDR X)))
               (REVERSE (CONS (CAR X)
                              (REVERSE (CDR (
                  REVERSE (CDR X)))))))))
```

Problemet här är att man måste tänka induktivt. Man måste uttrycka REVERSE i termer av de tillåtna LisP-funktionerna och REVERSE av en kortare lista. D.v.s. för att reversera (A B C D) så kan jag ta sista elementet i listan och CONSa det på omvändningen av de tre första elementen. Hur får jag sista elementet? Jo, genom att ta (CAR (REVERSE (CDR '(A B C D)))), här anropar vi alltså REVERSE rekursivt på en lista av längd 3, vilket är kortare än längden på (A B C D), vilken är 4. Hur får jag ut de tre första elementen då? Genom (REVERSE (CDR (REVERSE (CDR '(A B C D))))) får jag ut alla element utom det första och det sista, dvs (B C). Sedan CONSar jag på det första elementet igen och då har jag de tre första elementen. Även här har jag bara gjort REVERSE på kortare listor. Slutligen tar jag alltså omvändningen av de tre första elementen (också kortare än 4) och consar på det sista elementet, som jag redan har fått fram.

Detta kan verka invecklat, men knuten är att man kan anropa REVERSE rekursivt hur man vill - bara det är med en kortare lista. Till sist så anropar man REVERSE med en tom lista eller en lista av längd 1, och dessa enkla fall tar man hand om separat. Titta på Björns program och jämför med beskrivningen!



En alternativ lösning, gjord av Mathias Båge:

```
(DEFUN REVERSE (X)
  (COND ((NULL X) NIL)
        ((EQ (CDR X) T)
         (COND ((NULL (CDAR X)) (CAAR X))
               (T
                (REVERSE
                  (CONS (CONS (CONS (CADAR X)
                                    (CAAR X))
                              (CDDAR X))
                        T)))))
        (T (REVERSE (CONS (CONS NIL X) T)))))
```

Här används en annan teknik. Mathias använder sista CDR i listan som en flagga för att hålla reda på vad han gör.

3. Problemet: Skriv en funktion som gör samma sak som MEMQ, utan att den använder sitt eget namn för rekursionen.

Lösningsförslag:

```
(DEFUN MEMQ (X L)
  ((LAMBDA (X L F) (F X L F)) X L
   (QUOTE
    (LAMBDA (X L F)
      (COND ((NULL L) NIL)
            ((EQ X (CAR L)) T)
            (T (F X (CDR L) F)))))))
```

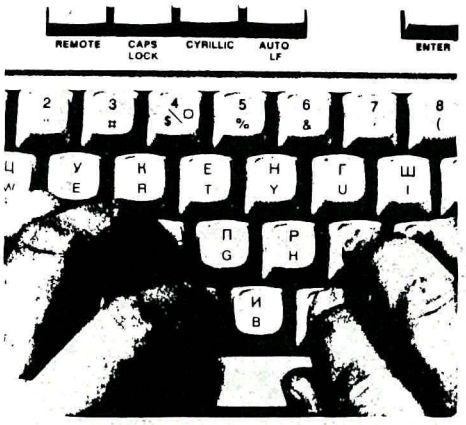
OBS! Detta program går inte att köra direkt i MacLisP, eftersom den LisPen inte har hela pure LisP som ett subset. Om man byter ut (F X ... mot (FUNCALL F X ... på de två ställen strängen förekommer, så går det bra.

/Lars-Henrik Eriksson

P.S. Svar på extrafrågan: Uppfinnar-Jockes program i STACKPOINTER numero 2-80, skulle sett ut så här (t.ex.):

```
(COND ((STACKP) (WRITE 'ARTICLE))
      ((FOODP) (EAT -1))
      (T (HACK 'AMIS)))
```

Tyvärr har inga svar influtit på denna fråga.



Visste du att...

Haga NADJA är smeknamn på ryska för...

Hageжga det ryska flicknamnet NADJEZJDA ...

som betyder "hopp"?

Bibliotek på STACKEN?

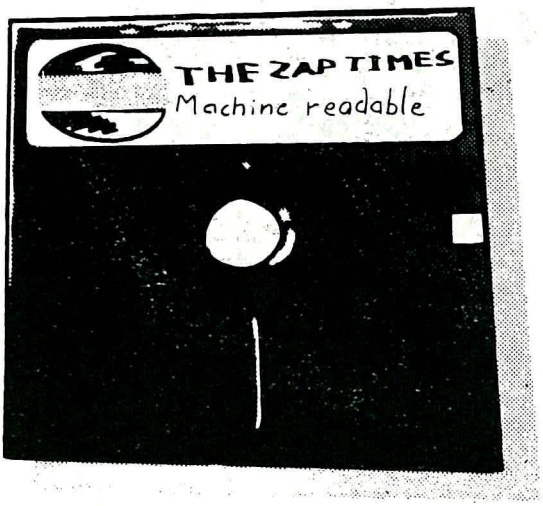
Just det. Stacken har ett visserligen blygsamt, men trevligt bibliotek bestående mest av kataloger och manualer, men även några fackböcker. Tyvärr är det dåligt med skönlitteratur, bidrag mottages tacksamt. Böckerna står i hyllan i 6502 ovanför kylskåpet.

Titta efter i hyllan om det finns något kul, sätt dig i stolen och läs! Lämna gärna kvar böckerna i lokalen, så blir det lättare för flera att läsa dom.

Stacken prenumererar också på några datortidningar, för tillfället BYTE och Dr. Dobbs Journal. Fler tidningar är på gång bl. a. Creative Computing ska införskaffas. Dessutom finns diverse osorterade tidningar samt tidningar från andra datorföreningar i Sverige.

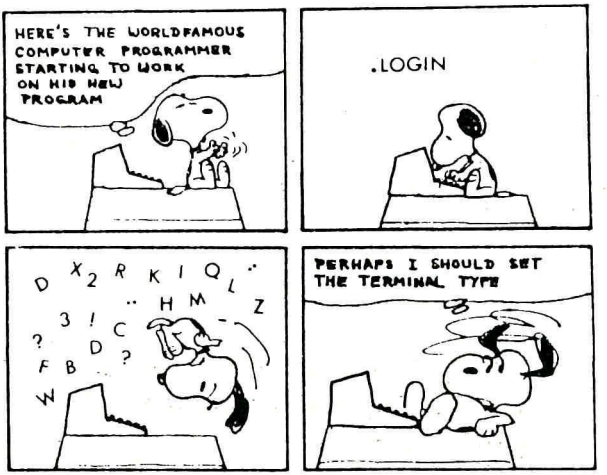
Om du har några förslag på vad som bör kompletteras så ta kontakt med mig.

Per Danielsson 1981-07-02 23:28:20



RUM SÖKES!

Stacken har ju som bekant en lokal på KTH, i samma våningsplan som NADAs terminalsalar. Denna lokal (kallad 6502) är emellertid ganska liten och trång. Vi har knappt plats med våra saker. Den är mest förvaringplats för Seven-S-datorn, tidningar och böcker. Det skulle antagligen bli bättre fart på hårdvaruaktiviteten i föreningen om vi finge en större lokal att vara i. Därför så efterlyser vi nu en FÖRENINGSLOKAL helst på KTH, men även annorstädes i Stockholm kan vi nog ta emot. Helst ska den vara hyresfri eller väldigt liten hyra, eftersom Stacken antagligen inte har råd att betala någon större hyra. Om DU vet om någon lämplig lokal så tveka inte, kontakta någon i styrelsen omedelbart, så kanske Stacken kan få sin efterlängtade lokal.



PROGRAM FÖR TMS 9900 PÅ NADJA.

Mitt intresse för mikroprocessorn TMS 9900 började för några år sedan då skolan jag gick i skaffade ett system. Det var för ett något bristfälligt system i början innan vi övertalade skolan att det inte gick att köra systemet med EN kassettbandspelare. Med två floppy diskar blev det enklare att använda systemet.

Fram till det att jag slutade skolan så blev det mest assembler programmering. Det enda högnivåspråk som fanns var BASIC och användes endast i nödfall. Jag hade bestämt mig för att bygga ett litet processorkort så jag skulle kunna köra hemma. Det är för tillfället tid som saknas för att plocka i hop det.

När jag fick veta att STACKEN hade fått en assembler för TMSen insåg jag genast att jag kunde använda den till. Möjligheten gavs mig att skriva mjukvaran på ett enkelt sätt. Asseblern var från Linköping och skriven i PASCAL. Då programmet inte gick att köra direkt på NADJA så blev det till att modifiera det.

Programmet fungerar och jag har inte hittat några fel. Jag är inte färdig än för det finns saker jag saknar och det kommer jag att införas allt efter jag får tid.

Under tiden jag inte kunde arbeta med modifieringarna av assemblern så började jag med en simulator. Simulatorn skulle jag ha för att kunna testa minna program. Två svårigheter har jag hittils träffat på.

Den första är att hitta en bra sammanfattning av status flaggornas funktioner. I en tunn bok med namnet "How to Build Your Own Working 16-Bit Microcomputer" hittade jag ett appendix där nästan allt vad jag ville veta fanns allt utom en flagga. I mitt sökande efter lösningen på problemet var jag i kontakt bl.a med Texas Instruments utan att hitta den. Men en STACKEN medlem kom med lösningen efter att ha testat på sitt eget system. Så nu kan arbetet fortsätta.

Den andra är att simulera I/O:n som kretsen använder då den är rätt ovanlig. All I/O är seriel. Det hela är uppbyggt så att man har 4096 bitar i en CRU (Communication Register Unit). Vid läsning av en bit i CRU så läggs adressen ut på adressbussen och en klockpuls (CRUCLK) visar när värdet på ledningen CRUOUT är rätt. Vid läsning så läggs adressen ut och läsningen får CRUIN triggas av en av de fyra klockfaserna.

Aldrig förr i historien har det varit möjligt att frambringa så många felaktiga resultat på så kort tid.

(Ur Lärobok I Algol, Studentlitteratur)

Då det finns fem instruktioner för I/O så är det ett kapitel för sig när man skall skriva simulatorn. Jag tror att jag skall göra så att hela CRU:n är ett 4096 bitars flagregister. All I/O skall göras genom externa operationer (XOP) som det finns 16 stycken. Sen gör jag helt enkelt så att simulatorn får veta vilken XOP som skall användas för vilken typ av I/O. Det kommer att finnas ett antal standard funktioner i simulatorn. Simulatorn får agera som en del av programmet man simulerar. Det är den enda lösning jag tror på. Tyvärr innebär detta att man inte kan testa I/O rutiner på simulatorn.

När simulatorn är klar skall den klara av att ladda objektkoden som assemblern tillverkar. Den skall klara av att disassemblera rader när man stegar igenom ett program eller när man själv vill se instruktionen i en viss position. I princip vara en monitor med olika funktioner.

Jag har även funderingar på att skriva en tiny-PASCAL för att sen skriva allt kraftfullare versioner av den. Man kanske kan få en användbar PASCAL till slut. Men detta ligger fortfarande på en hylla i hjärnkontoret, vem vet, sommaren är lång. Men först måste simulatorn bli färdig.

För den som är intresserad så ligger assemblern på TI9900.SFD på stackens konto på NADJA. Simulatorn kommer att läggas dit när den blir färdig. Om någon är intresserad av TMS 9900 och vill veta det mer om kretsen så har Texas Instruments givit ut en bok som heter "9900 Family System Design and Data Book" och innehåller allt man önskar veta och lite till. Det enda som saknas är de senaste kretsarna i början på 74LS600-serien (Refresh av dynamiska minnen, Memory mapper samt några andra kretsar).

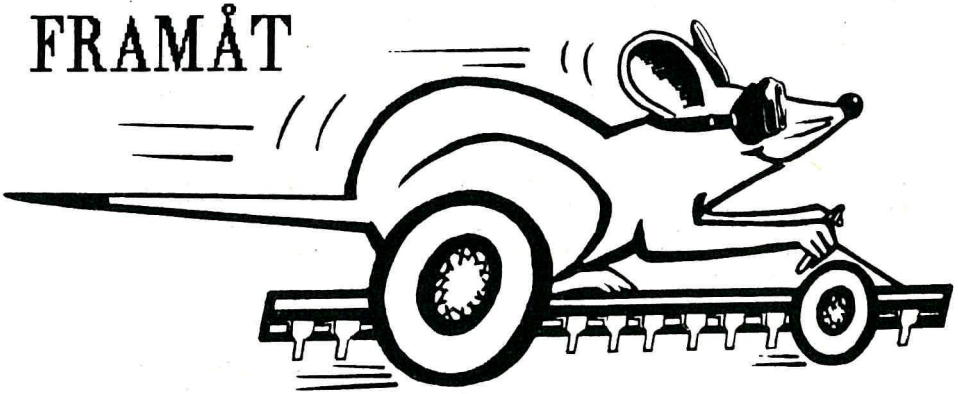
/Mats Jansson

LOLLY

Av Pete Hansen

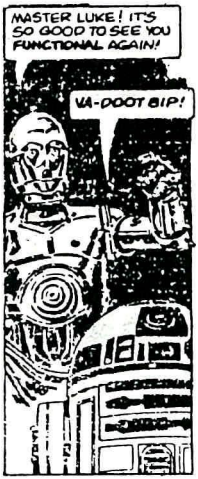
(Ur Expressen något datum)

16-BITS PROJEKTET GÅR FRAMÅT



Stackens 16:bitarsgrupp har nu bestämt sig för att välja både 68000 och Z8000 processorn där Z8000-biten utvecklas av S-Metric gänget. Undertecknad kommer definivt att välja 68000 pga att den är snabbare och har mer välstrukturerat instruktions-set (som antagligen är en följd av att den är mikroprogrammerad) samt har helt transparent addressering även med större addressrymder än 64Kbyte.

EDN (en amerikansk tidning) har låtit bla Motorola, Zilog och Intel koda ett antal standardalgoritmer (framtagna av Carnigie-Mellon University 1975) för minidatorer såsom interupthantering, strängsökning, bitmanipulering och quicksort, vilken gav som resultat att 10Mhz 68000 (som finns att köpa i USA men ej i Sverige än) var c:a 1.6 till 1.7 gånger snabbare än 6Mhz Z8000 och 1.7 till 2.0 gånger snabbare än 10Mhz 8086. De olika siffrorna beror på hur man beräknar medel. Den kompletta artikeln (med sourcelistor) finns tillgänglig i pärmen Intressanta Artiklar i stackens lokal. I sammanhanget bör nämnas att 10 Mhz versionen av 68000 kräver något snabbare minnen (c:a tio procent snabbare) än 6 Mhz versionen av Z8000. Med tanke på en tidigare artikel i SP kan nämnas att 68000 i "buggfritt" utförande nu funnits något mer än ett år på öppna marknaden.



Gruppens arbete har nu framskridit så långt att bussspecifikationen (systembussen och 68000 lokalbuss) nu är klar och (nästan) spikad. Vi räknar med att kunna sätta igång med kortlayouter för CPU-kort (68000), minneskort (som passar både till systembussen och 68000 lokalbuss, Z8000 har multiplexad lokalbuss så där passar det inte) och systembussinterface (för 68000 lokalbuss) inom en mycket snar framtid. Övriga kort som ingår i våra planer är ett MMU-kort till 68000 (när nu Motorolas MMU kommer, vi har dock vissa specifikationer på den) samt en floppycontroller avsedd att hängas på systembussen (ev med möjlighet att hängas även på 68000 lokalbuss). 16:bitarsgruppen har möte varje onsdag på D40 planet (samma plan som stackens lokal) kl 19.00. Alla intresserade är välkommna.

/Dan Norstedt

HACKERSLANG FRÅN A TILL Ö

I NADAs terminalsalar klockan 2000 en kväll. Några hackers sitter vid terminalerna och arbetar. Ytterligare en hacker kommer in.

Hacker 1: Hack, hack. State-of-the-world-p?
Hacker 2: Winning. Jag har nästan lyckats flusha en misfeature i det här programmet jag hackar. När jag är klar så blir det mycket cuspigare.
Hacker 3: Foodp?
Hacker 2: T! -1?
Hacker 1: NIL. Jag ligger i night mode, så jag har nyss ätit frukost.

Begrep du något? Om inte så skall du inte vara ledsen för det. Hackerslang är trassligt för den oinvigde, men för att göra kommunikation med hackers enklare presenterar STACKPOINTER nu en ordlista över hackerslang.

De förklarade orden utgör en del av den jargong som är vanlig bland NADA-hackers. Förklaringarna är bearbetningar av delar av innehållet i filerna JARGON.SWE<110,320,ORD> och GAM:HACKER.JAR på NADJA. Den förra innehållller uttryck som är specifika för NADA- och ELVIRA-hackers, den senare en stor sammanställning av amerikanska ord och uttryck. Om intresset för hacker-subkulturens språk skulle vara väckt, så ger dessa filer mycken värdefull information.

/Lars-Henrik Eriksson

Och här följer orden!

-P "-P"-konventionen. Man ställer en fråga genom genom att lägga på ett P efter ett ord och lägga parenteser runt hela frågan. Frågan besvaras med NIL för nej eller T för ja. Konventionen kommer ifrån LisP's konvention att ha -P på tester (t.ex. NUMBERP).
Exempel:
(FOODP) "Är du hungrig?"
(STATE-OF-THE-WORLD-P) "Hur är läget?"

-1 Universal default. Användes som standardsvar på frågor av typen: 'Var ska vi äta i dag då?'. I detta fall betyder svaret -1 "Restaurant China Garden".

17 Meta-uttryck. Betecknar Det Minst Slumpmässigt Valda Talet. Används då man vill beteckna ett tal, vilket som helst. Dess ursprung är höljt i dunkel.

4711 Används ibland i.st.f. 17, vid tillfällen då man vill beteckna ett lite större tal. Kommer troligen från namnet på "äkta" Eau-de-colonge, "No.4711"

6502 Namn på STACKENs lokal. Så kallad eftersom den, i likhet med mikroprocessorn med samma namn, har dåligt med utrymme för stacken.

11147 4711 i bas 8.

ANVÄNDARE En programmerare som tror på allt du säger till honom. Någon som ställer frågor, ofta triviala och irriterande. Ordet LOSER skrivs ibland LUSER, för att visa att användare (eng. USER) oftast är sådana.

AUTOMAGISKT Automatiskt, men på ett sätt som av jag av någon anledning (för att det är för invecklat eller för trivialt), inte har lust att beskriva. Se MAGISKT.

BAR Den andra metasyntaktiska variabeln. Se FOO. Exempel: "antag att vi har två funktioner FOO och BAR, FOO anropar BAR ...".

BUG Fel i ett program. Se FEATURE.

BURK Dator, terminal.

CDR (kudder) (från LisP) Att ta resten av något.

COBOL-FINGRAR fingrar som består av små korta stumpar, kapade ungefär vid första leden. Sjukdom som kommer av att programmera i COBOL, som är ett programmerings- språk där texten blir oerhört voluminös. COBOL-FINGRAR beror på att man nött ned fingrarna på tangentbordet.

CONSA (från LisP) Att sätta ihop något. "Om vi consar de här rutinerna så bör vi få ett fungerande program."

CUSPY — (av DEC-förkortningen CUSP, Commonly Used System Program) Något (t.ex. ett program) som är välgjort, trevligt att använda.

DET BJUDER VI PÅ — Mindre fel som man får stå ut med, som man inte orkar rätta.

DISABLA — Stänga av, stoppa, omöjliggöra.

DUNDERKLUMP — Benämning på en Teletype ASR33, urtypen för en skrivande terminal. Modellen är såld i n exemplar, och finns spridd över hela världen. Alla (tills helt nyligen) som sysslat med programmering har någon gång kommit i kontakt med DUNDERKLUMPEN. Den kännetecknas av sin stora vikt och höga ljudnivå. Har gett upphov till betecningen TTY som avser terminal.

EPSILON — En mycket liten mängd av något. Exempel: "Det krävs epsilon arbete att lägga in den här featuren." "Chansen att han skall hinna med sista tunneltåget är epsilon."

ERROR 220 — Fel som uppstår då man kopplar in elverket på datorbuss. Brukar ge upphov till en del intressanta ljusfenomen.

FAS — (hos människor) 1. Hur ens sömn och vakenperioder är orienterade i förhållande till den vanliga 24-timmarscykeln. En person som är ungefär 12 timmar ur fas sägs vara i "night mode". Uttrycken "day mode", "evening mode" och "morning mode" förekommer också. Exempel: "Vad ligger du i för fas?", "Jag gick upp 1600 och räknar med att gå hem runt 700." 2. "byta fas på det svåra sättet", hålla sig vaken under en lång tid för att komma i önskad fas. 3. "byta fas på det lätta sättet", att sova längre istället.

FEATURE — 1. En oväntad eller egendomlig egenskap hos ett program. Något som egentligen är fel, men inte strider mot någon dokumentation. "Det är inte en bugg, det är en feature!". 2. En välkänd och omtyckt egenskap. Exempel: "En feature i YSYTAT var att den slog av QZ-konverteringarna."

FLUSHA — Att ta bort något (onödigt, vanligtvis). "Jag tänker flusha all den här idiotiska koden."

FOO — 1. Den första metasyntaktiska variabeln. Används i exempel som namn på praktiskt taget allt, se BAR. 2. Förvånat utrop, oftast efter en obehaglig överraskning.

FORTRAN — FORmal TRAsh Notation. Fult ord. Skrives som dylika bör ofta som F-N.

FUDGEFAKTOR Ett värde eller en parameter som påverkar en process på ett (till synes) godtyckligt sätt. Exempel: "AMIS beräknar en fudgefaktor för att se om det lönar sig att...".

GARB Från LisP-världen. Från början förkortning för den s.k. Garbage Collectorn (GC) i LisP-system, vilken tar bort oanvända data, som då och då måste städas bort ur systemet för att inte ta onödig plats. Även namnet på LYSATORS klubbtidning. Att GARBA: ta bort saker och ting som ligger och skräpar. Värma upp GC, FOAuttryck för att generera mycket garbage.

GEB Förkortning för "Gödel, Escher, Bach: an Eternal Golden Braid", titeln på en bok av Douglas R. Hofstadter. Boken handlar om en massa olika saker, centrerat kring Gödels sats och AI. Den har blivit något av en kultbok bland hackers. Rekommenderas!

GREMLIN 1. Mytologiskt väsen, som förorsakade oförklarliga störningar i elektroniken i RAFs radarsystem under andra världskriget. 2. En SEMA i ALGOL 68. När man gör UP gremlins, kan det börja hända konstiga saker, särskilt med filer. Men det är faktiskt the gremlins som ser till att programmet fungerar ungefär som i ALGOL-68 rapporten, trots värddatorns ogästvänliga operativsystem. 3. Person, som har samma effekt på operativsystem, som en riktig gremlin. 4. En instans av 3) som brukar finnas på NADA.

HACK 1. Ursprungligen ett snabbt arbete som producerades vad som behövdes, men inte något mer. 2. Resultatet av ovannämnda arbete. 3. En "snygg hack", något listigt. 4. En något meningslös men vänlig hälsning. "Hack, hack".

HACKA 1. Att arbeta med något (vanligen ett program). Exempel: "Vad sysslar du med?", "Jag hackar MacLisP". 2. "Hacka sönder", ändra något på ett sådant sätt att det blir nära nog förstört.

HACKER 1. En programmerare som programmerar mycket snabbt eller skickligt. 2. En expert på ett speciellt program. Exempel "En LisP-hacker".

HACKVÄRDE Anledning att göra något till synes meningslöst bara för att resultatet är en hack. T.ex. så kan MacLisP läsa och skriva tal med romerska siffror. Koden för detta skrevs endast för sitt hackvärde.

HÅRIG — Onödigt invecklad, obegriplig. "Kommandona till CHANGE är otroligt håriga".

HAKMEM — MIT AI Memo 239 (Februari 1972). En klassisk samling eleganta matematik- och programmeringshackar från MIT och annorstädes. Ett exemplar finns att studera i STACKEN-lokalen.

JOJO-MOD — Ett tillstånd i vilket systemet snabbt alternerar mellan att vara nera och uppe. Exempel: "Hur går hackandet?", "Dåligt, maskinen är i jojomod."

KANONISK — Normaltillståndet eller beteendet hos någonting. Exempel: "Det kanoniska sättet att testa ett nytt program är att mata in så mycket skräp man kan i det och se vad som händer."

KLUDGE — En hack som avses lösa ett problem på ett snabbt, om än osnyggt sätt.

KRUNCHA — 1. Att bearbeta, vanligen på ett tidskrävande sätt. 2. Att återskapa assemblerkod ifrån ett objektprogram, vanligen i avsikt att ta reda på hur programmet fungerar, eller ändra det.

KUB — Avser alltid en s.k. Ungersk kub, matematiskt spel bestående av en kub med vridbara sidor. Ej att förväxla med fusk-, bluff- el. plagiatkuber, vilka är tillverkade i Taiwan.

LISP TERROR — Martin NILssons terroruttryck.

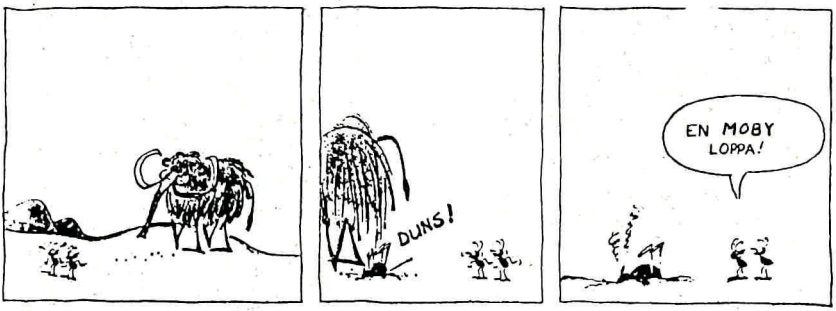
LOSE — Att misslyckas.

LOSER — Någon/något som misslyckas ofta genom att ha gjort på fel sätt från början.

MAGISKT — Något oförklarat eller med en förklaring som är för invecklad för att berättas.

MISFEATURE — En feature som ställer till mycket trassel. Det är inte detsamma som en bugg, då den kräver genomgripande förändringar i det inblandade systemet för att rättas till. "SFD-hanteringen i TOPS-10 är en verklig misfeature."

MOBY — 1. Stor, enorm eller komplex. 2. En adressrymd (256K) i en PDP-10 CPU. 3. Tilltalsord, ofta använd för att visa beundran, respekt och/eller vänskap för/med en kompetent hacker. "Godkväll, moby hacker, hur går programmet?"



MOON 1. Se MÅNE. 2. Dave Moon vid MIT.

MUNG (Mung Until No Good) Göra stora, oåterkallerliga ändringar, förstöra.

MÅNE En himlakropp, vars fas har stor betydelse för hackers.

MÅNENS FAS En slumpmässig parameter på vilket något sägs bero. Exempel: "Om det här fungerar beror på om man har foo-switchen satt och på månens fas"

N Ett godtyckligt (stort) tal. "Det finns n buggar i det här programmet!"

NIGHT MODE Se FAS.

NIL (från LisP-terminologi) 1. Nej. Används för att besvara frågor, i synnerhet frågor enligt "-P"-konventionen. Se T. 2. Välkänd NADA-hacker.

POM PHASE OF THE MOON, se MÅNENS FAS. Används oftast i uttrycket "POM-beroende", vilket betyder opålitlig.

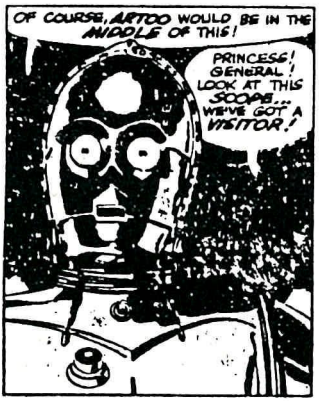
POPPA Av POP (stackoperation). Att återgå till en tidigare verksamhet.

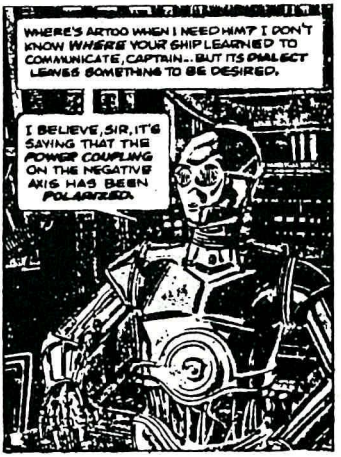
PROGRAMÄTAREN Ett litet djur som kryper omkring och avlägsnar alla program (speciellt källkod), som inte används på länge eller sparats på arkivmedium. Programätarens färg är grön. Den lilla gröna programätaren har varit framme igen.

PUSHA Av PUSH (stackoperation). Att avbryta en tidigare verksamhet i avsikt att fortsätta senare. Exempel: "Kan du inte pusha det där och följa med och äta?".

SLIPPA Att inte kunna eller få göra något. Ex. "Hur bär jag mig åt för att göra FOO?", "Du slipper!".

SLURPA Att helt och hållet läsa in någonting stort. Exempel: "Innan jag kan demonstrera programmet måste jag slurpa in de här filerna."





SMAK — Varianter, typ, sort. "DDT-kommandon finns i två smaker". Se VANILJ.

SNARFA — Att ta något i avsikt att använda det mer eller mindre utan upphovsmannens tillstånd.

STATE — Tillstånd, situation. "Vad har maskinen för state?", "Den är nere". "(STATEP)?", "Jag tror jag packar ihop och går hem".

SVART HÅL — Ett svart hål är en terminallinje som inte är kopplad någonstans. Uppträder vanligtvis i terminalväxlar, genom att man begär kontakt med en viss dator, och får en icke-existerande linje.

SÅ SYND — Ironiskt beklaga, t.ex: "Så synd, du slipper".

T — (från LisP-terminologi) Ja. Används för att besvara frågor, i synnerhet frågor enligt "-P"-konventionen. Se NIL.

TILT ERROR — Helt oväntat och/eller obegripligt fel, gärna med ett felmeddelande som inte säger någonting om vad som egentligen är orsaken. Ofta orsakat av att användaren har gjort något som får helt andra effekter än han tänkt sig.

TRAP — Avbrott i ett program, vanligen på grund av något fel. Används i överförd bemärkelse om hackers. Exempel: "Har du sett NIL?", "Han gick just men du har en chans att trappa honom om du skyndar dig".

TROLLKARL — (WIZARD) 1. Någon som vet hur ett komplext system fungerar, någon som kan lokalisera och rätta buggar i en nödsituation. 2. En person som har tillstånd att göra saker som är förbjudna vanliga dödliga. T.ex. köra ADVENTURE på dagtid.

TWENEX — (TWENty EXecutive) benämning på DEC-operativsystemet TOPS-20. Efter TENEX (TEN EXecutive), föregångare till TOPS-20 på ombyggda DEC-10 system.

VANILJ — Standardvariant. "Är det här din hackade TECO?", "Nej, det är vanilj-TECO".

VERKLIGHETEN — (THE REAL WORLD) 1. Plats där programmering nämns i samma andetag som uttryck i stil med FORTRAN, COBOL, IBM etc. 2. För programerare: plats där icke-programmerare förekommer eller där man sysslar med aktiviteter som inte har samband med programmering. 3. En värld där man går klädd i kavaj och slips och där arbetstid är definierat som 9 till 5.

VOIDA — (från ALGOL-68 terminologi) Kasta bort ett uträknat värde.

WIN — Att lyckas. SUPER-WIN, något jättebra. Se LOSE.

WINNER — En person som vet vad han gör och gör det bra.

WHEEL — Överstepräst på TWENEX-system (se dessa ord). Efter "WHEEL CAPABILITY" en bit i TWENEX-användares capability word som ger en möjlighet att göra allt.

ÖVERSTEPRÄST — En hacker som är systemprogrammerare och har makt över ett större datorsystem.

STACKENS STYRELSE 1981

Ordf Per Lindberg, skånegatan 68a 5 tr, 116 62 Sthlm
tel. 43 42 58

Jag går på Teknis, och jobbar extra som övningsass. på NADA. Läser gärna science fiction när jag inte programmerar.

Vice ordf Lars-Henrik Eriksson, Professorslingan 11/206
104 05 Sthlm, 15 46 67/ arb 018-15 54 00-1841

Jag är forskningsassistent i UPMAIL (forskningslabb för programmeringsmetodologi och AI i Uppsala). Jag har där arbetat med att driva institutionens DEC-20, men nu försöker jag äntligen slita mig loss för att börja min forskarutbildning. Året ut gör jag dock lumpen, bu. (I Stockholm, lyckligtvis).

Jag räknar Matematik som den Högsta av vetenskaperna och försöker därför göra Matematik av all datalogi jag får syn på. En del av resultatet kan beskådas i detta och föregående nummer av SP.

Sekr Robert Lindh, terapiv 16c 9 tr, 141 56 Huddinge
tel. 746 6031 / arb 749 9492

Jag är intresserad av bl.a. nya kretsar och mikroprocessorer.

Kassör Evald Koitsalu, skansbergsvägen 27, 141 70 Huddinge
tel. 46 70 11 / arb 97 00 90

Jag har sedan STACKEN startade förvaltat dess kassa.De primära uppgifterna har varit,att bokföra och uppdatera medlemmsregistret,samt se till att löpande utgifter blivit betalda, som tryckning av STACKPOINTERN,och inköp av övrig för föreningen behövligt material. Hur transaktionerna sköts kan kort beskrivas enligt följande: Styrelsen utser en medlem för att införskaffa UNIX till STACKEN. Han vänder sig då via telefon 08-467011 eller via brev till STACKEN Box 5079 140 05 HUDDINGE eller så tar han personligen kontakt med mig på SKANSBERGSVÄGEN27 i HUDDINGE.Av mig får han då ett förskott för att täcka kontanta utlägg för inköp av UNIX om tillräckligt med medel finns i kassan. Efter uträttat värv redovisar han inköpskvittot och eventuella vid köpet ej använda medel till mig.Redovisningen kan ske via post eller personligen. Härmed tror jag att det skall bli lite mera sprätt på klubbens ruljanser.

Kassörer Jan Michael Rynning, Klubbacken 36 3 tr,
126 56 Hägersten, 7878755 handledarrummet,
7877211 gamla forsknings-labbet, 468841 hem

Jag går på Teknisk Fysik, ungefär tredje året, jobbar på Numerisk Analys och Datalogi, som handledare, förser Stackens medlemmar med läsk, godis och annat kladd, skriver sökrutiner till AMIS, och tänker gå på alla intressanta möten och studiebesök i fortsättningen, nu när jag har muckat.

Intresserad av systemprogramvara, processor-arkitektur, texthantering, säkerhetsaspekter på datoranvändning, och en massa annat smått och gott.

Suppleant Hans Nordström, tingvallav 7f, 195 00 MÄRSTA
tel. 0760-167 40 / arb 0760-615 80

Är intresserad av mycket. I lagom portioner.

Redacteur Dag Rende, love almquist v 4a 3 tr, 112 53 Sthlm
tel. 50 30 93

Jag skall nog bli datalog någång, på riktigt alltså. För tillfället läser jag på matematikerlinjen. Det här med hårdvara är roligt, när det funkar. Jag är LISP-fantast, f.d. Z-80-fantast och intresserad av dator-typografi (fantast helt enkelt).

Revisorer Bill Önneflod, sätunav 14a, 195 00 Märsta
tel. 0760-173 38 / arb 34 00 20

Erik Forsberg, skebokvarnsv 281 1 tr, 124 35 Bandhagen
tel. 86 17 35

Lars Bengtsson, box 712, 181 07 Lidingö
tel. 60 34 15 / arb 767 6396

FORTRAN

anvandes pa egen risk !

STADGAR FÖR DATORFÖRENINGEN STACKEN

§ 1. FÖRENINGENS ÄNDAMÅL

Föreningen, vars namn är "Stacken", är en politiskt och religiöst obunden sammanslutning av personer som sysslar med datorexperiment och har till ändamål att tillvarata medlemmarnas gemensamma intressen och befrämja en sådan utveckling av verksamheten att den består och vidgas och att bland medlemmarna speciellt verka för ökade tekniska kunskaper inom mikrodatorområdet.

§ 2. MEDLEMMAR

Föreningens medlemmar är aktiva medlemmar och hedersmedlemmar.

§ 3. MEDLEMSKAP

Till hedersmedlem kan styrelsen kalla person som på utmärkt sätt tjänat föreningen och dess syften. Aktivt medlemskap kan beviljas fysisk person som erlagt av föreningsmötet fastställd medlemsavgift. Medlem som icke betalar årsavgiften inom föreskriven tid avförs ur medlemsregistret. Medlem som bryter mot föreningens stadgar, eller på annat sätt uppenbarligen skadar föreningen och dess syften, kan av styrelsen med enhälligt beslut uteslutas ur föreningen. Dessförinnan skall styrelsen dock bereda vederbörande tillfälle att avge förklaring. Utesluten medlem har besvärsrätt inför nästkommande årsmöte.

§ 4. RÖSTRÄTT

Hedersmedlem och aktiv medlem har vid föreningens allmänna sammanträden en röst. Rösträtt kan inte överlåtas. Styrelseledamot får ej nyttja egen röst för röstning och beslut i fråga om ansvarsfrihet för styrelsen för det gångna arbetsåret.

§ 5. AVGIFTER

Aktiva medlemmar erlägger årsavgift varje år vid månadsskiftet januari/februari. Avgiftens storlek för påföljande kalenderår bestäms av årsmötet. Beviljat medlemskap träder i kraft när fastställd avgift erlagts för löpande år. Vid beviljad ansökan om nytt medlemskap under kalenderårets tre sista månader gäller inbetald årsavgift även för efterföljande kalenderår.

§ 6. ORGAN

Föreningens organ är "Stackpointer" och föreningens ordförande dess ansvarige utgivare. "Stackpointer" skall vara ett medlemsblad för fortlöpande kontakt med medlemmarna och beröra de områden som kan vara av intresse med hänsyn till föreningens ändamål och verksamhet.

§ 7. ORGANISATION

Föreningens medlemmar samlade till allmänt sammanträde, vartill samtliga medlemmar kallats utgör föreningens högsta myndighet. Allmänna sammanträden är: årsmötet och extra sammanträden. Föreningens angelägenheter förvaltas av en inför årsmötet ansvarig styrelse.

§ 8. ÅRSMÖTET

a) Tidpunkt för och kallelse till årsmöte:

Årsmötet avhålls under april månad. Skriftlig kallelse till årsmötet skall tillställas medlemmarna under senast kända adresser och utsändas senast fjorton dagar före mötesdagen. Till kallelsen skall bifogas föredragningslista och avskrift av inkomna motioner.

b) Motioner:

Medlem som önskar få någon fråga behandlad vid föreningens årsmöte skall göra skriftlig anmälan härom. Sådan anmälan skall vara poststämplad eller styrelsen tillhanda senast den 31 januari och vara adresserad till styrelsen för att behandlas vid därpå följande årsmöte.

c) Årsmötesordning:

Årsmötet skall:
- Ta ställning till styrelsens redovisning för det gångna arbetsåret.
- Fastställa val av styrelseledamöter samt revisorer med suppleant.
- Förrätta val av ledamöter till styrelsevalberedningen för förberedande av nästa årsmötes val av styrelseledamöter samt revisorer med suppleant. Styrelsevalberedningen skall bestå av minst tre ledamöter och har till uppgift att avge förslag till kandidater för styrelsen. En av ledamöterna utses att vara sammankallande.
- Ta ställning till väckta motioner och framlagda styrelseförslag. Styrelsen skall till årsmötet avge yttrande till inkomna motioner.
- Besluta i stadgefrågor.

Ingen får samtidigt inneha mer än en av följande befattningar: styrelseledamot, revisor, ledamot i styrelsevalberedning eller suppleant eller vice för någon av dessa.

Vid årsmötet skall föras protokoll upptagande alla årsmötets beslut. Protokoll skall föreligga i justerat skick senast en månad efter mötets hållande och därefter snarast möjligt publiceras i föreningens organ. Vid årsmötet får endast ärenden avgöras som varit angivna i kallelse eller som står i omedelbart samband med sådana ärenden.

d) Föredragningslista till årsmötet:

1. Mötet öppnas.
2. Val av två personer att jämte ordförande justera mötesprotokollet. Justeringsmännen skall tillika tjänstegöra som rösträknare under mötet.
3. Val av ordförande för mötet.
4. Val av sekreterare för mötet.
5. Tillkännagivande av vid mötet uppgjord röstlängd.
6. Fråga om mötet är stadgeenligt utlyst.
7. Fråga om dagordningens godkännande.
8. Framläggande av styrelse- och kassaberättelse. I styrelseberättelsen skall även lämnas en redogörelse för resultatet av förra årets motioner.
9. Framläggande av revisionsberättelse.
10. Fråga om styrelsens ansvarsfrihet för det gångna arbetsåret.
11. Fastställa det tillkännagivna valresultatet för styrelseledamöter samt revisorer med suppleant.
12. Val av ledamöter till styrelsevalberedningen för nästa årsmötes val av styrelseledamöter samt revisorer med suppleant.
13. Behandling av inkomna motioner. Styrelsen skall avge yttrande till inkomna motioner. Motioner numreras 13:1, 13:2 o.s.v.
14. Behandling av styrelseförslag. Förslagen numreras 14:1, 14:2 o.s.v.
15. Behandling av budget för innevarande år.
16. Fastställande av medlemsavgifter för året efter det under vilket årsmötet hålls.
17. Synpunkter på verksamheten för innevarande år.

§ 9. EXTRA SAMMANTRÄDE

Extra sammanträde med föreningen skall äga rum minst två gånger per termin. Beträffande kallelse och protokoll gäller vad som i § 8 stadgas för årsmötet. Vid extra sammanträde får endast ärenden avgöras som varit angivna i kallelsen eller som står i omedelbart samband med sådana ärenden.

§ 10. BESLUT

I de fall inte annorlunda framgår av dessa stadgar skall vid föreningens allmänna sammanträden och styrelsesammantäden fattas beslut med enkel majoritet. Omröstning sker öppet om inte annat begärs eller stadgas. Vid lika röstetal vid öppen omröstning har mötesordförande utslagsröst och vid sluten omröstning avgör lotten.

§ 11. STYRELSEN

Styrelsen skall:
- aktivt verka för att ändamålen i § 1 i möjligaste mån uppfylls,
- förvalta föreningens tillgångar och redovisa dessa inför årsmötet,
- utge föreningens organ,
- besluta i frågor rörande inträde i och uteslutande av medlem ur föreningen,
- föra föreningens talan inför myndigheter,
- låta arrangera föreningens tävlingar,
- uppehålla erforderlig kontakt med motsvarande organisationer i Sverige och andra länder,
- i övrigt på eget initiativ handla i föreningens anda och i enlighet med dessa stadgar.

Vid samtliga styrelsesammanträden skall föras protokoll upptagande alla styrelsens beslut. Protokoll skall föreliga i justerat skick senast en månad efter sammanträdets hållande och bör inom två veckor i kopia tillställas samtliga styrelseledamöter.

Styrelsen består av:
- ordförande,
- vice ordförande,
- sekreterare,
- kassaförvaltare,
- hexmästare,
- suppleant,
- redaktör för föreningens organ.

Styrelsen sammanträder på kallelse av ordförande eller vid förfall för denne av vice ordförande, eller då minst tre styrelseledamöter gör framställning härom. För att styrelsen skall vara beslutsmässig erfodras att fyra styrelseledamöter är närvarande. Styrelsen kan till sin hjälp kalla eller anställa personer för särskilda uppdrag.

§ 12. ERSÄTTARE VID AVGÅNG

Inträffar den situation att vice ledamot eller suppleant inte finns, om styrelseledamot, revisor eller ledamot i styrelsevalberedningen avgår före valperiodens utgång, eller om utsedd suppleant avgår före valperiodens utgång, skall styrelsen utse en ersättare för den återstående valperioden utan att eljest föreskrivna valordning iakttas.

§ 13. RÄKENSKAP OCH REVISION

För föreningen skall räkenskapsåret sammanfalla med kalenderåret. Bokslutet skall vara verkställt senast den 15 februari, då räkenskaperna jämte tillhörande handlingar, ävensom av styrelsen förvaltningsberättelse jämte vinst- och förlusträkning samt balansräkning för senaste räkenskapsåret, skall för granskning överlämnas till de utsedda revisorerna. Dessa skall granska föreningens räkenskaper och förvaltning och ta del av styrelsens och verkställande utskottets protokoll, kontrollera vården av föreningens fasta och lösa egendom samt i övrigt kontrollera att av styrelsen fattade beslut och vidtagna åtgärder är i enlighet med föreningens stadgar samt senast den 1 mars över granskningen avge skriftligt utlåtande, revisionsberättelse, vari ansvarsfrihet för styrelsen bestämt till- eller avstyrkts. Innan eventuell anmärkning bereda styrelsen eller ledamot därav, som anmärkningen avser, att avge yttrande i saken.

§ 14. TECKNINGSRÄTT

Föreningen tecknas av dess ordförande och kassör var för sig.

§ 15. STADGEÄNDRINGAR

Beslut om ändringar av dessa stadgar är inte giltig med mindre än att enhälligt beslut därom fattats av årmötet, eller om beslut därom fattats på två på varandra följande allmänna sammanträden, varav ett årsmöte och därvid på båda sammanträdena biträtts av minst 3/4 av de röstande, samt att det mellan de båda sammanträdena skall ha förflutit minst 60 dagar. Dessutom skall ändringsförslaget utförligen ha omnämts i kallelsen till sammanträdet eller sammanträdena i fråga.

§ 16. FÖRENINGENS UPPLÖSNING

Beslut om upplösning av föreningen är inte giltigt med mindre än att därom fattas beslut på två på varandra följande allmänna sammanträden, varav ett årsmöte, och därvid på båda sammanträdena biträtts av minst 4/5 av de röstande, samt att det mellan de båda sammanträderna skall ha förflutit minst 90 dagar. Dessutom skall förslaget till upplösning av föreningen utförligen ha omnämts i kallelsen till sammanträderna i fråga. Vad som efter gäldande av föreningens skulder återstår av föreningens tillgångar vid en eventuell upplösning skall tillfalla ändamål som överensstämmer med föreningens syften och som beslutats vid det sista sammanträdet.

Dragon Curve.
Ett exempel på en rekursivt definierad kurva.
Nedan: LISP-programmet som ritade kurvan.

```
(defun dragon (depth size)
  (prog (x-pos y-pos)
        (moveto (setq x-pos (quotient size 3))
                (setq y-pos (quotient size 3)))
        (plot-dragon (quotient size (expt 2 n)) 0 depth 'left)))

(defun plot-dragon (x y depth parity)
  (cond ((zerop depth)
         (drawto (setq x-pos (plus x-pos x))
                 (setq y-pos (plus y-pos y))))
        (t (setq depth (sub1 depth))
           (caseq parity
                  (right (plot-dragon x y depth 'left)
                         (plot-dragon y (minus x) depth 'right)
                         (plot-dragon x y depth 'left)
                         (plot-dragon (minus y) x depth 'right))
                  (left  (plot-dragon (minus y) x depth 'left)
                         (plot-dragon x y depth 'right)
                         (plot-dragon y (minus x) depth 'left)
                         (plot-dragon x y depth 'right))))))
```


I WANT YOU
for the
GALACTIC
EMPIRE