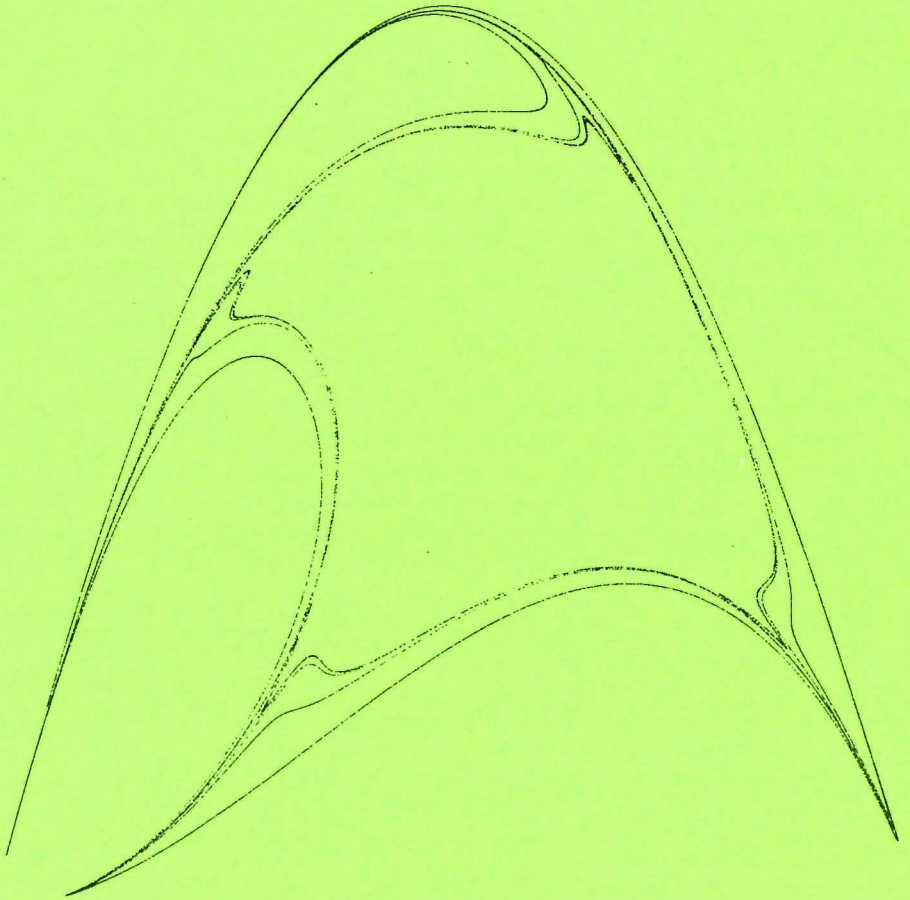
STACKPOINTER

2-1986. Organ för Datorföreningen STACKEN, KTH.



Datorföreningen STACKEN

STACKEN är datorföreningen på KTH.

Vi har en mängd intressanta verksamheter på gång. Dock hänger det ytterst på den enskilde medlemmen att avgöra vad han eller hon vill göra för föreningen. STACKEN är en ideell förening, där intresse för datorer är den gemensamma faktorn. Sedan föreningen grundades 1978 har vi (bland annat) åstadkommit:

- samköp av mikrodatorbyggsatser
- diverse kurser och föredrag
- studiebesök på intressanta ställen
- AMIS, den portabla EMACS-kompatibla editorn för TOPS-10, VMS, PRIME, NORD, ...
- STACKPOINTER, föreningens tidning, som numera utkommer rätt ofta
- en egen datorhall för våra stordatorer
- en egen DEC-10:a (se nedan)

Då detta skrivs (februari 1986) har vi sedan ett par månader en egen DEC-10, som vi installerat, felsökt och kör på. Det är en gammal modell med KA10-processor. Vi kallar henne KATIA. Hon står i vår nya klubblokal och maskinhall "B30", på Brinellvägen 30 (V-sektionen) på gaveln mot Lill-Jansskogen (där vi har en egen ingång). En våning ovanför finns en hörsal, där vi håller till vid större möten.

Ordinarie möten är första torsdag i varje månad kl. 19 i antingen lokalen eller hörsalen. Är Du intresserad av föreningen, är Du välkommen till något av våra möten. Vill Du sedan bli medlem, skickar Du in en skriftlig ansökan till vår postadress:

Datorföreningen STACKEN
c/o NADA
KTH
100 44 STOCKHOLM

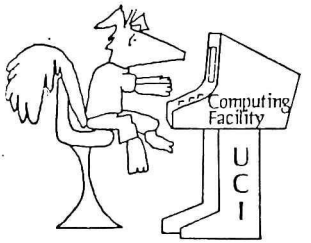
⋆ *Computers have more fun!* ⋆
⋆ *Computare necesse est!* ⋆

Per Lindberg

STACKPOINTER

STACKPOINTER är organ för Datorföreningen STACKEN. STACKPOINTER utkommer när material i tillräcklig mängd finns, förhoppningsvis 4–5 gånger per år. Återgivande av delar av innehållet är tillåtet när källan anges.

Ansvarig utgivare:	Mats O Jansson
Redaktör:	Hans Nordström
I redaktionen:	Jan Michael Rynning
Färdigställd:	1986-04-17



UCI Computing Facility

NEWSLETTER

VOLUME 18, NUMBER 2 UC IRVINE COMPUTING FACILITY WINTER 1986

Blateratus Rectoris

This summer a very significant change marked the UCI Computing Facility: the departure of the Sigma-7 and the DECsystem-10. These computers arrived at UCI in the summer of 1969 and served UCI faculty and students for fifteen years. Their departure marked the end of an era and the loss of memorabilia for many of us.

The Sigma-7 was sold. The DECsystem-10 was sold for junk, but escaped the ignominious fate of the scrap heap, and therein lies a love story...

The cast...

Mike Iglesias has been around the Computing Facility for more than ten years including his undergraduate days. He has played a pivotal role in the Facility and the Facility has played a pivotal role in Michael. (It is where he met the woman he would marry, but that is not part of this love story.) Michael has worked for a lot of years as the system guru for the -10.

But good love stories seldom involve only one person...

The other major actor in this story is Chuck Catlett. Chuck manages the UCI Computing Facility. He has managed the Facility for so long he can do it in his sleep.

The story...

We knew we had to get rid of the DECsystem-10 in the summer of 1984. We advertised the system to the outside world. We received an offer from a junk dealer to buy the equipment for $3,000. It was the only offer we got.

Michael could not accept that the object of his worry and care for the past just-short-of-a-decade was to be sold to the junk man. Michael sent a note over the Arpanet that UC Irvine had a DECsystem-10 for sale. From Stockholm came the answer to Michael's electronic ad: we want it.

A computer club in Stockholm offered to buy our DECsystem-10 for $3,100. They owned several DEC computers (even older than ours) and this was a fantastic opportunity to move their collection into a slightly more modern era.

Michael then asked Chuck if this deal could be done.

Chuck said no, too much trouble to sell the system to someone in Stockholm. The -10 had to be quickly dispatched. The Facility needed the space.

Michael had the -10 disassembled and put in various parts of the Computer Science Building. No one would notice.

Michael set out to get an export license.

Weeks passed...

Michael struggled to obtain an export license. Concern increased as the corners now occupied by the DECsystem-10 were needed for other things. It seemed each day that parts of the -10 had to be moved to make way for incoming paraphernalia. Chuck kept threatening Michael with the specter of the junk dealer.

The check for $3,100 arrived from Stockholm. Chuck wouldn't let it be cashed.

Days passed... The export license arrived.

Folks at Lawrence Livermore Laboratory called Chuck and recounted that they had sold some equipment to a computer club in Stockholm, and were told to send it to UCI for final shipment to Sweden. Chuck, with visions dancing in his head of thousands of pieces of equipment arriving at Irvine from throughout the country, moved slowly, decidedly over that line which separates the sane from the insane. No telling how large a budget the computer club had.

Chuck, really excited, tells the Livermore folks not to ship their equipment to Irvine under any circumstances. Michael frantically sends electronic messages to Stockholm.

Days pass...

...The equipment from Livermore arrives at Irvine.

Pressure for the space now occupied by the -10 builds. The Livermore equipment is at the Receiving Department.

Creating hope in the minds of a few, Michael gets a message that someone from Stockholm is coming to UCI to pack the equipment.

One morning the phone rings. I make a mistake and answer it. It is Peter in Stockholm.

Peter: "I can't locate you on the map."

Peter speaks better English than I do Swedish, fortunately, but the conversation isn't going very well. "Irvine" is a mystery to him.

"Have you heard of John Wayne?", I ask.

Peter: "The cowboy?"

"Yes", I said, "Fly to the cowboy."

"Yes", Peter answered and the phone went dead. Would we ever see Peter?

...Days pass.

Peter showed up a day after a forty-foot cargo container arrived at our loading dock. Peter stayed with us a couple of days, packing his precious cargo from Livermore and Irvine in the container, along with several cases of Coke.

We cashed the check.

...Weeks pass.

Michael got an electronic message from Sweden, the -10 had arrived in Rotterdam, its next stop Stockholm.

...Postscript

Michael received an electronic note this week from Peter; the shipping container had arrived in Stockholm. The DEC-10 is currently computing at its new home.

David Sheldon
Director of Computing
and Telecommunication

AMIS-manualen: slutrapport

Den länge otåligt emotsedda AMIS-manualen, avfattad å engelska språket, är nu fullbordad. Visserligen ej ännu sidindelad, "layoutad", tryckt och mångfaldigad, men i alla fall skriven, strippad på SCRIBE-koder samt uppdaterad till att motsvara rådande status på den produkt den beskriver, dvs AMIS.

Jo, det har tagit tid. Främst på grund av diverse programvarustrul — påminn mig om att aldrig lita till prefabricerade layoutsprogram. Dock skulle jag vilja påpeka att dröjsmålet har haft sina fördelar. När jag började med manualen var det nämligen väldigt många AMIS-funktioner som "var på gång", som "skulle skrivas nästa vecka", som var "praktiskt taget färdiga" och som därför skulle tas med i manualen. Nåja. Det hör till charmen med hackers att vi är oförbätterliga optimister. Hade manualen blivit klar parallellt med det då pågående arbetet på AMIS, hade den innefattat mycket som senare visade sig ogenomförbart, hävdat existensen av mycket som aldrig blev av, och därigenom blivit starkt otillförlitlig, dvs en dålig manual.

I det här sammanhanget skulle jag — eftersom jag nu har ordet — vilja berätta litet om hur det går till att göra en manual. Det påpekades nämligen under arbetets gång, att jag inte borde vänta mig någon assistans från annat håll, om jag ville få något gjort, utan att jag måste vara beredd på att göra allting själv, inklusive testning, faktagranskning, layout och tryckning. Ingen annan kunde förväntas vara intresserad. Detta förföll mig visserligen något märkligt, då jag inte kunde tänka mig att jag vore den enda som ville ha en AMIS-manual, men i och för sig förstår jag idén bakom synpunkten — om jag inte litade till någon annan, skulle jag kanske lägga den egna manken till i större utsträckning och äntligen bli färdig någon gång.

AMISAMISAMISAMISAMISAMIS

Nu är det bara så, att en enda person inte kan framställa en fungerande manual. När manualen är skriven, måste den a) korrekturläsas, b) faktagranskas. Det senare involverar ofta testning av överensstämmelse mellan programmet och manualen - om nu inte faktagranskaren har hela källkoden i huvudet utantill förstås, men detta kan och ska man inte utgå ifrån. Synpunkten att jag, såsom manualförfattare, själv skulle korrekturläsa och faktagranska manualen, är

tyvärr helt ohållbar. En sådan manual skulle jag aldrig lämna ifrån mig. Teoretiskt sett funnes det kanske en möjlighet på miljonen att manualen trots detta skulle vara användbar, men med överväldigande sannolikhet skulle den inte duga till annat än dasspapper. Ska den vara dels läsbar, dels tillförlitlig, MÅSTE NÅGON ANNAN än manualförfattaren ha gått igenom den, annars är den helt enkelt inte godtagbar. Vanligtvis anlitar man olika korrekturläsare för språklig och saklig granskning, eftersom det ännu så länge är ont om humaniker, dvs folk som sitter inne med båda typerna av färdigheter. Själv hade jag emellertid den otroliga turen att få tillgång till en granskare med såväl tillräckliga språkkunskaper som fullt adekvat hackerfärdighet. Manualen är alltså både språk- och faktagranskad (tack, Elric!) och därför fullt publicerbar med avseende på innehållet.

Manualen är alltså skriven, korrläst och uppdaterad till AMIS' nuvarande nivå. Detta är allt jag hinner med, så i det skicket överlämnar jag härmed dokumentet till STACKENs styrelse. TEXning eller annan layout samt tryckning ser jag mig nödsakad att överlåta på någon annan därtill hågad. Likaledes tvivlar jag på att jag kommer att hinna med några omfattande uppdateringar när (om?) det blir dags för AMIS version 2 att slå omvärlden med häpnad.

Med varm tass,

/EvA Albertsson

AMISAMISAMISAMISAMISA

Vårmötesprotokoll

Protokoll fört vid Datorföreningen STACKENs vårmöte, avhållet torsdagen 1986-02-06 klockan 19 i sal V4 på KTH.

Närvarande medlemmar i alfabetisk ordning:
Olle Betzén, Bengt Bäverman, Ulf Börjeson, Henning Croona, Hans Davidson, Anders Ekström, Sven Erik Enblom, Lars-Henrik Eriksson, Per Eriksson, Leif Eurén, Bengt Forsgren, Mats O Jansson, Stellan Lagerström, Björn Levin, Per Lindberg, Lars S Ljungdahl, Peter Löthberg, Kurt Minnberg, Thord Nilsson, Åke Nordin, Hans Nordström, Bertil Nyström, Tomas Nyström, Jan Michael Rynning, Åke Sjögren, Per Sjöholm, Bernhard Stockman, Jörgen Städje, Urban Sundström, Peter Svanberg, Erik Svensson, Hugo Westerlund, Klaus Zeuge och Bill Önneflod.

§1. Mötets öppnande.
Stellan Lagerström hälsade alla välkomna och förklarade mötet öppnat.

§2. Val av justeringsmän.
Sven Erik Enblom och Bernhard Stockman valdes till justeringsmän.

§3. Val av mötesordförande.
Stellan Lagerström valdes till mötesordförande.

§4. Val av mötessekreterare.
Mats O Jansson valdes till mötessekreterare.

§5. Tillkännagivande av uppgjord röstlängd.
De närvarande sade sina namn och prickades av på röstlängden.

§6. Frågan om mötet är stadgeenligt utlyst.
Mötet ansåg sig vara stadgeenligt utlyst. Per Lindberg påpekade att det i fortsättningen borde stå i kallelsen att mötet hålls i hörsal V4 och inte i B30, STACKENs lokal i våningen under, som det stått den här gången. Anslag om att mötet hade flyttats hade satts upp utanför B30.

§7. Fråga om dagordningens godkännande.
Jan Michael Rynning förklarade att ordet "eventuellt" i §12–§15 betydde att de bara behövde behandlas om mötet ansåg det nödvändigt och att anledningen till att de fanns med var att Jonas Mölsä avsagt sig uppdraget som ordförande.

§8. Verksamhetberättelse.
STACKEN har installerat och dragit igång en egen stordator, en DEC-10, som vi kallar för Katia. Under våren och sommaren har vi varit sysselsatta

med att skaffa fram och lägga in datorgolv i den lokal KTH tilldelat oss att ha som datorhall, samt tiggt ihop och installerat elmateriel, kylanläggning, m m. I början på hösten (närmare bestämt den 14e oktober, klockan 4:37 på morgonen) fick vi igång datorn. Arbetet avslutades med en stor invigningsfest den 23e oktober, med runt 120 deltagare, och med att Rektor Gunnar Brodin invigde datorhallen och datorn. Till invigningsfesten kom bl a besökare från datorföreningen Studio 54 vid Norges Tekniska Høgskole i Trondheim, datorföreningen Lysator vid Linköpings Tekniska Högskola och Medicindata vid Göteborgs Universitet. Representanter från Chalmers Datorförening och från den nybildade datorföreningen vid Lunds Tekniska Högskola hade besökt oss tidigare, under installationsarbetet.

Under året har STACKEN gjort tre studieresor: till Göteborg, Åbo och Linköping. Det var Chalmers Datorförening som hade bjudit ner oss till Göteborg, för att deltaga i firandet av deras 10-årsjubiléum. När vi ändå var där passade vi även på att besöka Medicindata vid Göteborgs Universitet, som har en DEC-10-anläggning. Besöket i Åbo var ett svarsbesök hos INFÅ, datorföreningen vid Åbo Akademi, som bl a ordnade studiebesök åt oss på DAVA, Finlands tredje största datorföreteg. I Linköping hälsade vi på hos Lysator, LiTHs datorförening, samt gjorde studiebesök hos Context Vision, ett avknoppningsföretag från LiTH som specialiserat sig på avancerad bildbehandling.

På DECUS Nordic-konferensen i Trondheim och DECUS Europekonferensen i Cannes var STACKEN representerad av ett flertal medlemmar, som bevakade föreningens intressen. Efter konferensen i Trondheim passade STACKENs ordförande Jan Michael Rynning på att besöka Studio 54 på NTH. Vid konferensen i Cannes utsågs Jan Michael Rynning till redaktör för DEC-10/20-SIGens tidning, DECsystem-10/DECSYSTEM-20 EuroCopy.

I slutet på året tog STACKEN hem en DEC-10-dator till, som är modernare än den vi har och som vi fick köpa till skrotpris från University of California. Merparten av den utrustningen har nu konverterats från 110 V/60 Hz till 220 V/50 Hz och satts igång.

Det mesta av det som hänt under året har rapporterats i STACKPOINTER, som utkommit med sju nummer, varav ett dubbelnummer. Förutom STACKPOINTER har det gjorts utskick inför studieresorna till Göteborg och Åbo.

§9. Revisionsberättelse.

Båda revisorerna rekommenderade ansvarsfrihet. Lars S Ljungdahl ansåg det lämpligt att STACKEN anlitar en revisionsbyrå för föreningens ekonomi. Mötet beslöt att så skulle ske. Se även bilagor.

§10. Balansräkning.
Se bilaga.

§11. Ansvarsfrihet för avgående styrelse.
Mötet beviljade den avgående styrelsen ansvarsfrihet.

§12. Ev. val av styrelsemedlemmar.
Matti Rendahl valdes till ny ordförande.

§13. Ev. fastställande av firmatecknare.
Matti Rendahl utsågs att, jämte Mats O Jansson, enligt STACKENs stadgar teckna föreningen var för sig.

§14. Ev. val av revisorer.
Mötet ansåg inte att frågan behövde behandlas.

§15. Ev. val av valberedning.
Mötet ansåg inte att frågan behövde behandlas.

§16. Mötesdagar under 1986.
Arbetsmöte hålls varje torsdag. Månadsmöte hålls första torsdagen i varje månad (oavsett om den är helgdag). Höstmöte hålls torsdagen 1986-12-04. Alla möten hålls i B30, STACKENs lokal på Brinellvägen 32, eller i hörsal V4 i våningen ovanför.

§17. Motioner.
Jan Michael Rynning valdes enhälligt till hedersmedlem.

§18. Övriga frågor.
Tomas Nyström har tiggt till sig ett CPU-kort till kortläsaren.
Björn Levin har förlängt kontona på Kicki och Katia.
Medlemar som betalt medlemsavgiften mer än en gång tar kontakt med Björn Levin eller Mats O Jansson.
Lars S Ljungdahl frågade om hur det blir med slipsarna. Styrelsen svarade att man arbetade på problemet.
Peter Löthberg bad folk att tigga byggmatriel till B30. Han har även skaffat fram ritningar till PDP-15.
Per Lindberg åtog sig att skriva ett litet program för [66,66], som skulle fånga in intresserade personer.
Den stora plottern skall flyttas från Elmer med bil på nästa månadsmöte.
Tomas Nyström skall titta på programvaran i de Tandbergterminaler vi fått.

§19. Mötets avslutande.
Stellan Lagerström förklarade mötet avslutat.

OBS! Protokollet är ännu inte justerat. *Mats O Jansson*

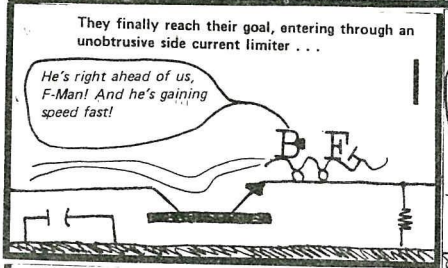
FORTRAN MAN

Lee Schneider

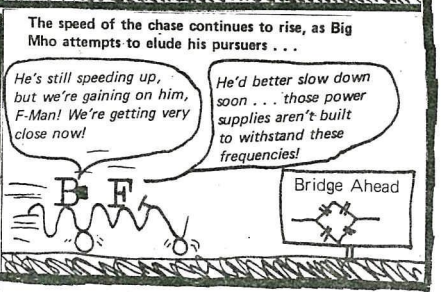
Todd Voros

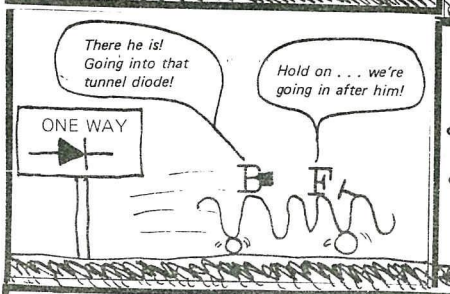
Fortran Man and Billy Basic speed past low-core, which is a fortress of protection against the majority of the outside world . . .

They finally reach their goal, entering through an unobtrusive side current limiter . . .



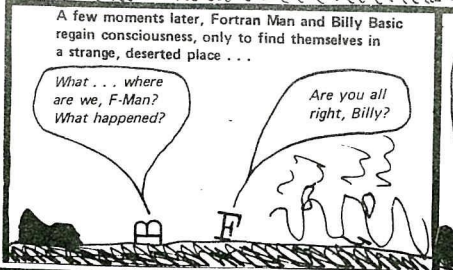
The speed of the chase continues to rise, as Big Mho attempts to elude his pursuers . . .



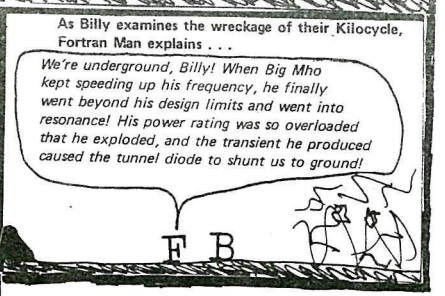


They enter the tunnel diode, when suddenly . . .

A few moments later, Fortran Man and Billy Basic regain consciousness, only to find themselves in a strange, deserted place . . .



As Billy examines the wreckage of their Kilocycle, Fortran Man explains . . .



TO BE CONTINUED...

Ut mot verkligheten

Exjobb Exjobb

VME eller MULTIBUS II?

Den frågan ställer sig många systembyggare idag. Leverantörerna av respektive busskoncept nöjer sig inte med att förhärliga sig själva, utan man tar till moderna politikergrepp genom att kasta sk-t på varandra. Långa rader av artiklar skrivs men resultaten tycks mer bero på författarens övertygelse än på busspecens innehåll.

Så vad skall en vanlig dödlig göra? Ett bra förslag är att prova själv, att kasta sig ut i praktiken och se vad som händer! Och det hade jag tänkt göra genom att bygga ett CPU-kort resp ett muliportsminne för att få erfarenhet av vilka dolda svårigheter som finns.

Men till detta behöver jag hjälp! Jag tror att det skulle passa precis för två exjobbare att göra dessa konstruktioner. Det finns en del skisser och tankar att utgå från. För avlusning av byggena finns en utvecklingsmiljö bestående av en VME-dator med Versados samt assembler och C-kompilator.

Jobbet är alltså en kombination av hård- och mjukvara där program måste fram för att visa att konstruktionerna fungerar.

Administrativt kan jobbet ses som två helt separata eller ett dubbeldito. Det hela skall i alla händelser ske i AGEMA:s lokaler mitt emot Danderyds gymnasium med start så snart som möjligt för den som vill bli klar före semestern.

AGEMA? Jo, det är nya namnet på gamla AGA Infrared där flera av initiativtagarna till STACKEN jobbade år 0. Vi pysslar med värmekameror och alltmer sofistikerade bildutvärderingssystem. Våra grejor används till forskning och diverse felsökning typ medicin, koll av el-installation, värmeläckage osv. Utpräglat civila områden alltså.

Lyft inte ögonbrynen bara, utan ta hela telefonluren och ring mig

Tel: 08/753 34 00 09.00-16.30
08/753 34 11 16.30-18.00
0762/116 23 18.30-08.30

eller skriv till AGEMA, Box 3, 182 11 DANDERYD

AGEMA Infrared Systems AB
Utvecklingsavdelningen

Dennis Engström

Dennis Engström

The ULTIMATE Language

by Lindmark & Mölsä

I. **Kort presentation av språket.**
II. **Språkets syntax i BNF-notation.**
III. **ULTIMATE-systemets uppbyggnad.**
IV. **Några algoritmer skrivna i ULTIMATE.**
V. **Körexempel.**
VI. **Programlistor.**

I. Kort presentation av ULTIMATE.

Idéerna till ULTIMATE är hämtade ifrån Dijkstra: *A Discipline of Programming*, kap. 4. Dock har vi byggt idéerna om post- och preoperatorer på C:s motsvarigheter.

Språket är väldigt litet, det finns inte ens någon möjlighet att definiera egna procedurer eller funktioner. Det har endast fyra fördefinierade procedurer (för in- och utmatning) och tre styrstrukturer (sekvens, val och slinga).

Operationer på data är definierat för enkla variabler av heltalstyp och logisk typ, men pga. att implementationen är gjord i Lisp kan man tilldela variablerna nästan vad som helst, men då är det bara in- och utmatning som är definierat. Det går alltså inte att bearbeta variabler av andra typer.

Både styrstrukturen för val och den för slinga är uppbyggd kring begreppet *guarded command set*. Ett *guarded command set* består av en ordnad mängd *guarded commands*, där varje *guarded command* består av ett villkor (*guard*) och en följd av satser. Villkoret ser till så att följden av satser inte exekveras om villkoret inte är uppfyllt.

Styrstrukturen för val ser ut så här:

IF *guarded command set* FI

och fungerar som så att *guarded command set* gås igenom tills man hittar ett villkor som är sant. Då utförs följden av satser som hör till det villkoret. Kvarvarande *guarded commands* ignoreras. Om inget av villkoren är sant är detta ekvivalent med den tomma satsen. Val fungerar alltså som Simulas IF-THEN-ELSE sats.

Styrstrukturen för slinga ser ut så här:

DO *guarded command set* OD

där genomgången av villkor UPPREPAS (med medföljande exekvering av tillhörande satser) ända tills det för varje *guarded command* gäller att motsvarande villkor är falskt.

```
Ex.  DO x > y -> x := x - y
       ! y > x -> y := y - x
     OD
```

Detta är Euklides algoritm för att hitta största gemensamma delare till två tal x och y (utropstecknet skiljer

guarded commands åt — *guarded command separator*). Villkoret för uthopp ur slingan är att x = y (= SGD av ursprungliga x och y).

Båda dessa styrstrukturer (val och slinga) är sammansatta satser och kan därför nästlas godtyckligt i varandra. Styrstrukturen sekvens skrivs som vanligt genom att man radar upp sats efter sats (med kolon emellan).

Idéen med post- och preoperatorer illustreras bäst med några exempel:

Preplus: i += 7	Först adderas 7 till i och sen lämnar i ifrån sig sitt värde.
Postplus: i =+ 7	i lämnar först ifrån sig sitt värde och sen (innan värdet används) adderas 7 till i.
i := 1 : i := 5 + (i =+ 7)	i lämnar ifrån sig värdet 1 (tilldelningen har nu formen i := 5 + 1), därefter adderas 7 till i (så i=8) och sist utförs tilldelningen i := 5 + 1 (så i=6).
i =: 7	lämnar först ifrån sig i:s gamla värde och sen sätts i till 7 (tvärtemot i:= 7).
a =: (b =: a)	gör att två variabler byter värde med varandra, "swap". På liknande sätt kan man "rotera" värdena på de tre variablerna a,b,c med a =: (b =: (c =: a)).

Det går även bra att skriva t ex a := (b := x) eller a := b + (c := 5) men *inte* a := b := 7.

Styrstrukturen för slinga tillsammans med post- och preoperatorer gör det möjligt att på ett enkelt sätt skriva generella for-slingor.

Ex. i := 0 : j := 0 :
DO (i += 1) ≤ 15 AND (j += 7) ≤ 60 ->
...
OD

Kommentarsatser inleds med dollartecken '$' och sträcker sig till efterföljande kolon. Kommentarerna sållas bort av kompilatorn och påverkar inte objektkoden på minsta sätt.

Den tomma satsen betecknas SKIP.

II. Definition av ULTIMATE.

Definitionen är på en variant av BNF notation, där:

::=	betyder definieras som.
(*mummel*)	betyder att det som står inom parenteser får upprepas en eller flera gånger.
<*mummel*>	betyder att det som står inom hakar måste tas med.

	betyder att man måste ta med ett av de alternativ som åtskiljs av lodstrecket.
'mummel'	betyder exakt de tecken som står inom apostrofer.
MUMMEL	ord skrivna med versaler kan inte definieras ytterligare utan ska skrivas som de står (= 'mummel').
...	betyder fortsätt sekvensen (på uppenbart sätt).
expr	är en förkortning för *expression.*
aexpr	är en förkortning för *arithmetic expression.*
lexpr	är en förkortning för *logical expression.*
aop	är en förkortning för *arithmetic operator.*
lop	är en förkortning för *logical operator.*
relop	är en förkortning för *relational operator.*

Här är definitionen av ULTIMATE:

program ::=
statement-list

statement-list ::=
statement (':' *statement*)

statement ::=
assignment | *if-statement* | *do-statement* | *procedure* | *comment* | SKIP

assignment ::=
identifier < *pre-op* | *post-op* > *expr*

identifier ::=
letter (*letter* | *numeral* | '_')

letter ::=
'a' | ... | 'z' | 'A' | ... | 'Z'

numeral ::=
digit (*digit*)

digit ::=
'0' | ... | '9'

expr ::=
aexpr | *lexpr*

aexpr ::=
< '+' | '−' > *aexpr* | '(' *aexpr* ')' | *aexpr aop aexpr* | *identifier* | *numeral*

aop ::=
'+' | '−' | '*' | '/' | '^' | *pre-op* | *post-op*

pre-op ::=
':=' | '+=' | '−=' | '*=' | '/='

post-op ::=
'=:' | '=+' | '=−' | '=*' | '=/'

lexpr ::=
'(' *lexpr* ')' | *lexpr lop lexpr* | *aexpr relop aexpr* |
NOT < TRUE | FALSE | *identifier* | '('*lexpr* ')' > |
identifier | TRUE | FALSE

lop ::=
AND | OR

relop ::=
'<' | '<=' | '>' | '>=' | '=' | '<>'

if-statement ::=
IF *guarded-command-set* FI

guarded-command-set ::=
guarded-command ('!' *guarded-command*)

guarded-command ::=
guard '−>' *statement-list*

guard ::=
lexpr

do-statement ::=
DO *guarded-command-set* OD

procedure ::=
NEWLINE |
IN '(' *identifier* (*identifier*) ')' |
OUT '(' *identifier* | '(' *expr* ')' (*identifier* | '(' *expr* ')') ')' |
BLANKS '(' *aexpr* ')'

comment ::=
$ (*letter* | *numeral* | *harmless-character*)

harmless-character ::=
'!' | '$' | '%' | '?' | '+' | '−' | '^' | '~' | '@' | '*' | '_'

III. ULTIMATE-systemets uppbyggnad.

Ett program består av en lista där satserna åtskiljs med kolon. Programmet kan antingen skapas direkt på fil, eller matas in med hjälp av Lisp-funktionen ENTER. Om det skapas på fil måste filen ha utseendet
(setq programnamn '(sats1 : sats2 : ...))

Listan bearbetas sedan av LISTIFY, som lämnar en lista där varje sats i sin tur är en lista. Därefter översätter LISPIFY varje (listifierad) sats till en Lisp-funktion. Det kompilerade programmet består då av en lista av listor, där varje dellista är en Lisp-funktion. Körning av det kompilerade programmet görs med RUN, som gör EVAL på varje dellista.

För att underlätta har vi definierat funktionerna COMPile
```
(DEFUN COMP (PROGRAM)
    (LISPIFY (LISTIFY PROGRAM)))
```
och EXecute
```
(DEFUN EX (PROGRAM)
    (RUN (LISPIFY (LISTIFY PROGRAM)))).
```

Innan ULTIMATE-systemet kan köras måste man ladda in vissa funktioner och göra en del initieringar. Detta görs med
(lload init)
(init)

Nedan följer en kort beskrivning av de funktioner som används av ULTIMATE-systemet.

AOPS	Definierar macron för post- och preoperatorer.
COMPILE	Kompilerar ett ULTIMATE-program.
CONSTINIT	Initierar TRUE och FALSE till T respektive NIL.
DOOD	Gör om en DO-OD konstruktion till en DO-sats där avbrottsvillkoret är den COND-sats man får om man låtsas att det är en IF-FI konstruktion (se ex.).
ENTER	Läser in programrad efter programrad med hjälp av READ i en slinga.
EXECUTE	Kompilerar och kör ett ULTIMATE-program.
IFFI	Gör om en IF-FI konstruktion till en COND-sats (se exempel).
IN	Läser in värden till en eller flera varibler med hjälp av READ i en slinga.
INFTOPRE INFAUX INFITER	Omvandlar ett uttryck i infixnotation till motsvarande uttryck i prefixnotation. (Klarar även AND OR NOT och unärt plus och minus.) Används i LISPIFY.

INIT	Initierar systemet (laddar nödv. filer etc.).
LISPIFY	Omvandlar en "listifierad" ULTIMATE-sats till en Lisp-funktion med hjälp av COND.
LISTIFY	Gör listor av alla satser och villkor. Åtskillnadstecknen ':', '!' och '–>' behövs därför inte längre.
NOTEQUAL	Definierar funktionen <> arg1 arg2.
OPRINIT	Initierar operatorernas prioritet bl a.
OUT	Skriver ut ett eller flera uttryck på skärmen med hjälp av PRINC.
RUN	Kör ett kompilerat program.

Pedagogiska exempel:

```
kod:            IF x > y –> x := x – y
                 ! y > x –> y := y – x
                FI
efter listify:  (IFFI ((x > y) (x := x – y))
                      ((y > x) (y := y – x)) )
efter lispify:  (COND ((< x y) (SETQ x (– x y)))
                     ((< y x) (SETQ y (– y x))))
kod:            DO x > y –> x := x – y
                  ! y > x –> y := y – x
                OD
efter listify:  (DOOD ((x > y) (x := x – y))
                      ((y > x) (y := y – x)) )
efter lispify:  (DO ()
                    ((NOT (COND ((< x y) (SETQ x (– x y)))
                                ((< y x) (SETQ y (– y x))))))) )
```

Om det (efter listify) står IFFI eller DOOD som nyckelord, anropas (i lispify) Lisp-funktionen iffi resp. dood som översätter konstruktionen till ren Lisp-kod. (Med Lisp-funktion förstås i det här fallet att funktionen är skriven i Lisp (av oss, inte av MacLisp-systemet)).

Begränsningar i implementationen:

- Man bör inte använda variabelnamn som redan är definierade och viktiga i Lisp.
- '/' (delat med) måste skrivas som '//'.
- Man måste ha mellanslag mellan varje "item".
  Ex: a␣:=␣3␣;␣IF␣a␣<>␣3␣–>␣g␣+=␣7␣FI

- Syntaxkontrollen är i det närmaste obefintlig.

Interna fulheter:

- Det sätt som vi använder Lisps 'cond' och 'do' på, för att implementera våra IF- och DO-satser, gör att vi får problem med bieffekter. För att undvika dessa bieffekter lägger vi sist i varje *guarded command* in satsen SKIP, vilket är en sann styggelse.

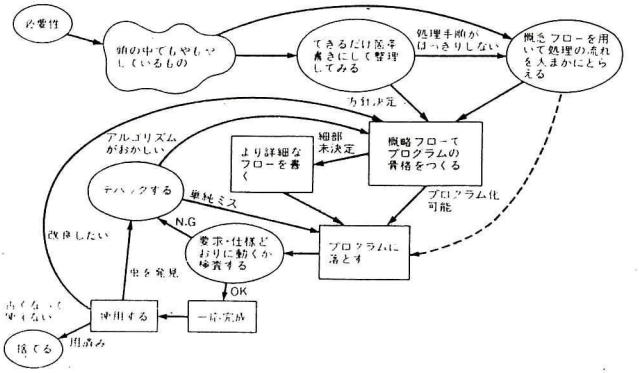
Schyssta features:

- Variabler initieras inte automagiskt vilket medför att om man är så korkad så att man använder variabler innan de har fått nåt värde så åker man dit.
- Om programmet kraschar kastas man ut i Lisp-systemet, som är tillräckligt kraftfullt för att klara av alla tänkbara situationer (vilket i och för sig inte hjälper en person som bara programmerat i ULTIMATE hela sitt liv, men men ...).

★★★ THIS IS NOT A BUG ★★★
★★★ IT'S A FEATURE ★★★

forts kommer

<図 6-1>
プログラムの一生



forts här

IV. Några algoritmer skrivna i ULTIMATE.

i) Beräkning av primtal.
Programmet beräknar alla primtal upp till en given gräns genom att kolla delbarhet med alla udda tal upp till och med roten ur talet.

```
Newline :                                   $ Ser snyggt ut          :
Out ("Beräknar primtal. Ge övre gräns! ") :
In (limit) : Out ("2") :                    $ 2 är primtal           :
number := 1 :
DO number += 2 ≤ limit ->
  a := 0 :                                  $ Börja beräkna roten    :
  c := 1 :
  DO c ^ 2 ≤ number -> c =* 2 OD :
  DO c <> 1 ->
    c =/ 2 :
    IF (a + c) ^ 2 ≤ number -> a += c FI
  OD :                                      $ Kvadratroten uträknad  :
  root := a : odd := 1 :
  prime := TRUE :
  DO (odd += 2) ≤ root AND prime ->
    IF number / odd * odd = number -> prime := FALSE FI :
  OD :
  IF prime -> Out (number) FI :             $ Skriv ut primtalet     :
OD :
Newline
```

ii) Sortering av fyra tal.
Programmet sorterar fyra tal i storleksordning på ett verkligt elegant sätt. (Eventuellt kan programmet byggas ut för att klara betydligt fler tal.)

```
Newline :
Out ("Ge fyra tal som du är lite osäker på storleksordningen av! ") :
Newline :
In (q1 q2 q3 q4) :
DO q1 > q2 -> slask := q1 : q1 := q2 : q2 := slask
   ! q2 > q3 -> slask := q2 : q2 := q3 : q3 := slask
   ! q3 > q4 -> slask := q3 : q3 := q4 : q4 := slask
OD :
Out ("Samma tal i storleksordning är" q1 q2 q3 q4) :
Newline :
```

V. Körexempel.

Här nedan följer ett körexempel. Efter man har gjort INIT är det bara att sätta igång och programmera.

```
@macliSP.EXE.5

Welcome to MacLisp at Vera.
NIL
(lload init)

;Loading LEDIT 39
;Loading SUBFORK 50
;Loading DEFVSY 83
;Loading EXTSTR 90
;Loading DEFMAX 98
ÄLLOAD of file PS:<JONAS-MOLSA.ULTIMATE>INIT.LSP.3 completed.Å*
(init)


;Loading BACKQ 53
;Loading DEFMACRO 165
;Loading MACAID 116
;Loading CNVD 2
ÄLLOAD of file PS:<JONAS-MOLSA.ULTIMATE>AOPS.LSP.1 completed.Å
ÄLLOAD of file PS:<JONAS-MOLSA.ULTIMATE>ENTER.LSP.1 completed.Å
ÄLLOAD of file PS:<JONAS-MOLSA.ULTIMATE>EXECUTE.LSP.1 completed.Å
ÄLLOAD of file PS:<JONAS-MOLSA.ULTIMATE>INFTOPRE.LSP.1 completed.Å
ÄLLOAD of file PS:<JONAS-MOLSA.ULTIMATE>IO.LSP.1 completed.Å
ÄLLOAD of file PS:<JONAS-MOLSA.ULTIMATE>LISPIFY.LSP.1 completed.Å
ÄLLOAD of file PS:<JONAS-MOLSA.ULTIMATE>LISTIFY.LSP.1 completed.Å
ÄLLOAD of file PS:<JONAS-MOLSA.ULTIMATE>OPRINIT.LSP.2 completed.Å
ÄLLOAD of file PS:<JONAS-MOLSA.ULTIMATE>SUPPORT.LSP.3 completed.Å
NIL

(setq hejtest (enter))

OK, fläska in programmet. Avsluta med QUIT.

i := 0 :
do (i += 1) <= 5 -> out("hej") od
quit
(I := 0 : DO (I += 1) <= 5 -> OUT ("hej") OD NEWLINE)
(ex hejtest)

hej hej hej hej hej
T
```

VI. Programlistor.

All filer ligger på VERA::PS:<JONAS-MOLSA.ULTIMATE>, för dem som vill prova.

```
(defun enter ()       ;Make a list of statements. (Not a list of lists.)
  (terpri)
  (princ '"OK, fläska in programmet. Avsluta med QUIT.")
  (terpri) (terpri)
  (do ((part (read)) (program nil))
      ((equal part 'quit)
       (append program (list 'NEWLINE)))     ;Always end program with a
    (setq program (append program(list part))) ;  NEWLINE. Looks good when
    (setq part(read))))                        ;  executing.



(defun init ()
  (terpri)
  (mapcar 'lload
          '((aops)
            (enter) (execute)
            (inftopre) (io)
            (lispify) (listify)
            (oprinit)
            (support)
            ))
  (constinit)   ; defined in support
  (oprinit)
  (terpri))



(defun constinit ()
  (setq true t)
  (setq false nil))


(defun opcode (op)
  (get op 'operator))


(defun skip ()
  (let ((v t))
    (setq v t)))


(defun dood (list)
  `(do () ((not ,(cons 'cond list)) T) ))


(defun iffi (list)
  (cons 'cond list))
```

```
(defun oprinit ()
  (defprop bottom  t operator)
  (defprop =:  t operator)
  (defprop :=  t operator)
  (defprop =+  t operator)
  (defprop +=  t operator)
  (defprop =-  t operator)
  (defprop -=  t operator)
  (defprop =*  t operator)
  (defprop *=  t operator)
  (defprop =//  t operator)
  (defprop //=  t operator)
  (defprop OR  t operator)
  (defprop AND  t operator)
  (defprop NOT  t operator)
  (defprop >  t operator)
  (defprop >=  t operator)
  (defprop <  t operator)
  (defprop <=  t operator)
  (defprop =  t operator)
  (defprop <>  t operator)
  (defprop +  t operator)
  (defprop -  t operator)
  (defprop *  t operator)
  (defprop //  t operator)
  (defprop ^  t operator)

  (defprop bottom  -1 prio)
  (defprop =:   1 prio)
  (defprop :=   1 prio)
  (defprop =+   1 prio)
  (defprop +=   1 prio)
  (defprop =-   1 prio)
  (defprop -=   1 prio)
  (defprop =*   1 prio)
  (defprop *=   1 prio)
  (defprop =//  1 prio)
  (defprop //=  1 prio)
  (defprop OR   2 prio)
  (defprop AND  3 prio)
  (defprop NOT  4 prio)
  (defprop >    5 prio)
  (defprop >=   5 prio)
  (defprop <    5 prio)
  (defprop <=   5 prio)
  (defprop =    5 prio)
  (defprop <>   5 prio)
  (defprop +    6 prio)
  (defprop -    6 prio)
  (defprop *    7 prio)
  (defprop //   7 prio)
  (defprop ^    8 prio))

(defun <> (arg1 arg2)
  (not (equal arg1 arg2)))

(defun ^ (arg1 arg2)
 (expt arg1 arg2))
```

```
(defmacro := (x y)  `(setq ,x ,y) )

(defmacro =: (x y) `(prog1 ,x  (setq ,x ,y))  )

(defmacro += (x y) `(setq  ,x  (+ ,x ,y))  )

(defmacro =+ (x y) `(prog1 ,x  (setq ,x (+ ,x ,y))))

(defmacro -= (x y) `(setq  ,x  (- ,x ,y))  )

(defmacro =- (x y) `(prog1 ,x  (setq ,x (- ,x ,y))))

(defmacro *= (x y) `(setq  ,x  (* ,x ,y))  )

(defmacro =* (x y) `(prog1 ,x  (setq ,x (* ,x ,y))))

(defmacro //= (x y) `(setq  ,x  (// ,x ,y))  )

(defmacro =// (x y) `(prog1 ,x  (setq ,x (// ,x ,y))))
```

```
(defun listify (program)      ; Makes a list of each statement.
  (do ((newpgm '() )
       (tmp)       )
      ((null program) newpgm)                ; While exists stmts in program do
    (setq tmp (lfy_stmt program))            ;   Listify first stmt in program
    (setq newpgm                             ;   Insert listified stmt last -
          (append newpgm (list (car tmp))))  ;     in newpgm
    (setq program (cdr tmp))  ))             ;   Pop listified stmt off program
                                             ; Return newpgm

(defun lfy_stmt (program)     ; Listify statement.
  (let (tmp1 tmp2)
    (cond ((or (equal (car program) 'In)       ; This cond determines what to
               (equal (car program) 'Out)      ;  do for each type of statement.
               (equal (car program) 'Blanks))
           (append (list (list (car program) (cadr program))) (cddr program)))
          ((equal (car program) 'Newline)
           (append (list (list (car program))) (cdr program)))
          ((equal (car program) '$)
           (lfy_unit program))
          ((equal (car program) 'SKIP)
           (append (list (list (car program))) (cdr program)))
          ((equal (car program) ':)
           (append (list (list 'SKIP)) (cdr program)))
          ((equal (car program) 'DO)
           (setq tmp1 (lfy_gcs (cdr program)))
           (setq tmp2 (append '(DOOD) (car tmp1)))
           (cons tmp2 (cdr tmp1)))
          ((equal (car program) 'IF)
           (setq tmp1 (lfy_gcs (cdr program)))
           (setq tmp2 (append '(IFFI) (car tmp1)))
           (cons tmp2 (cdr tmp1)))
          ((opcode (cadr program))
           (lfy_unit program))
          )))
```

```
(defun lfy_gcs (program)                ; Listify guarded command set.
  (do (tmp1 tmp2 guard                  ; The DO-loop takes one gcm at a time.
            (gcs '() ))
      ((or (equal (car program) 'OD)    ; Return from loop when you hit OD
           (equal (car program) 'FI))   ;  or FI.
       (cons gcs (cdr program)))
    (setq tmp1 (lfy_unit program))
    (cond ((and (null (cdar tmp1)) (atom (caar tmp1))) ; Is the guard an ident-
           (setq guard (caar tmp1)))                   ; fier or a constant ?
          (t (setq guard (car tmp1))))                 ; No => it's an expr.
    (setq program (cddr tmp1))          ; Listify the statements after the guard.
    (setq tmp2 (lfy_gcm program))
    (setq tmp1 (car tmp2))
    (setq tmp1 (append tmp1 '((SKIP))))
    (setq gcs (append gcs (list (cons guard tmp1))))
    (setq program (cdr tmp2))  ))


(defun lfy_gcm (program)                ; Listify guarded command.
  (do (tmp
       (gcm '()) )
      ((or (equal (car program) '!)     ; Guarded command separator.
           (equal (car program) 'OD)    ; Guarded command terminator.
           (equal (car program) 'FI))   ; Guarded command terminator.
       (cond ((equal (car program) '!)   (cons gcm (cdr program)))
             ((equal (car program) 'OD)  (cons gcm program))
             ((equal (car program) 'FI)  (cons gcm program))  ))
    (setq tmp (lfy_stmt program))
    (setq gcm (append gcm (list (car tmp))))
    (setq program (cdr tmp)) ))


(defun lfy_unit (program)       ; Listify guard or statement.
  (do ((stmt '() ))
      ((or (equal program nil)
           (equal (car program) ':)
           (equal (car program) '!)
           (equal (car program) '->)
           (equal (car program) 'OD)
           (equal (car program) 'FI) )
       (do () ((not (equal (car program) ':) ))  ; Get rid of excessive colons.
         (setq program (cdr program)))
       (cons stmt program) )
    (setq stmt (append stmt (list (car program))))
    (setq program (cdr program)) ))
```

```
(defun lispify (program)           ; Makes a list of LISP-lists.
  (do ((newpgm '())
       (part program (cdr part)))  ; Compile one statement per round.
      ((null part) newpgm)
    (cond ((atom (car part))
           (setq newpgm (append newpgm (list (car part)))))
          ((equal (caar part) '$) t) ; '$' signals a comment, which
                                     ; should be ignored.
          ((or (equal (caar part) 'NOT)  ; Unary operators.
               (equal (caar part) '+)
               (equal (caar part) '-))
           (setq newpgm (append newpgm (list (inftopre (car part))))))
          ((opcode (cadar part))
           (setq newpgm (append newpgm (list (inftopre (car part))))))
          ((equal (caar part) 'DOOD)
           (setq newpgm
                 (append newpgm
                         (list (dood (mapcar 'lispify (cdar part)))))))
          ((equal (caar part) 'IFFI)
           (setq newpgm
                 (append newpgm
                         (list (iffi (mapcar 'lispify (cdar part)))))))
          ((equal (caar part) 'Newline)
           (setq newpgm (append newpgm (list '(newline)))))
          ((equal (caar part) 'SKIP)
           (setq newpgm (append newpgm (list '(skip)))))
          ((equal (caar part) 'BLANKS)
           (setq newpgm
                 (append newpgm (list `(blanks ,(inftopre (cdar part)))))))
          ((equal (caar part) 'IN)
           (setq newpgm (append newpgm (in (cdar part)))))
          ((equal (caar part) 'OUT)
           (setq newpgm (append newpgm (out (cdar part)))))
          (t (princ (car part))  (princ '( - not a statement ))))))



(defun inftopre (ae)
  (cond ((atom ae) ae)                 ; Easy case first,
        (t (infaux ae nil nil))))      ; else stacks start empty.


(defun infaux (ae operators  operands)
  (cond ((not (opcode (car ae) ))                  ; Operand ?
         (infiter (cdr ae)                         ; No, work on CDR...
                  operators
                  (cons (inftopre (car ae))        ; after recursion on CAR.
                        operands)))
        ((equal (car ae 'NOT))
         (infiter (cdr ae)
                  operators
                  (cons
                   (list 'NOT (inftopre (cadr ae)))
                   operands)))
        (t    ; Hopefully unary plus or minus, put in a zero first and restart.
         (infaux (cons '0 ae) operators operands))))
```

```
(defun infiter (ae operators operands)
  (cond ((and (null ae) (null operators))            ; Finished ?
         (car operands))
        ((and (not (null ae))
              (or (null operators)
                  ( > (get (car ae) 'prio)           ; Compare priorities of
                      (get (car operators) 'prio)))) ; operator and list head.
         (infaux (cdr ae)
                 (cons (car ae) operators)           ; Push operator,
                 operands))                          ;  and continue.
        (t (infiter ae
                    (cdr operators)                  ; Pop operator,
                    (cons (list (car operators)
                                (cadr operands)      ; construct sublist,
                                (car operands))
                          (cddr operands))))))       ; and pop operands.

(defun in (varlist)
  (do ((part (car varlist) (cdr part)) (in nil))
      ((null part) in)
    (setq in (append in
                     (list `(setq ,(car part) (read) ))))))


(defun out (arglist)
  (do ((part (car arglist) (cdr part)) (out nil))
      ((null part) out)
    (setq out (append out
                      `((princ ,(inftopre (car part)))(princ 'ö ö))))))
                               ; ^^^^^^^^ argument might be an expression.


(defun newline ()
  (terpri) )

(defun blanks (ntimes)
  (do ((n ntimes (sub1 n)))
      ((equal n 0) t)
    (princ 'ö ö)))


(defun run (program)        ; Runs a compiled program.
  (mapcar 'eval program)
  t)


(defun execute (program)    ; Executes a program.
  (run (lispify (listify program))))


(defun ex (program)  ; 'Cause I'm lazy.
  (execute program))


(defun compile (program)         ; Compiles a program made with ENTER.
  (lispify (listify program)))
```

Framsidan

Om man sätter x och λ till små positiva värden (0.00001 eller så) och sedan gång på gång går igenom sekvensen

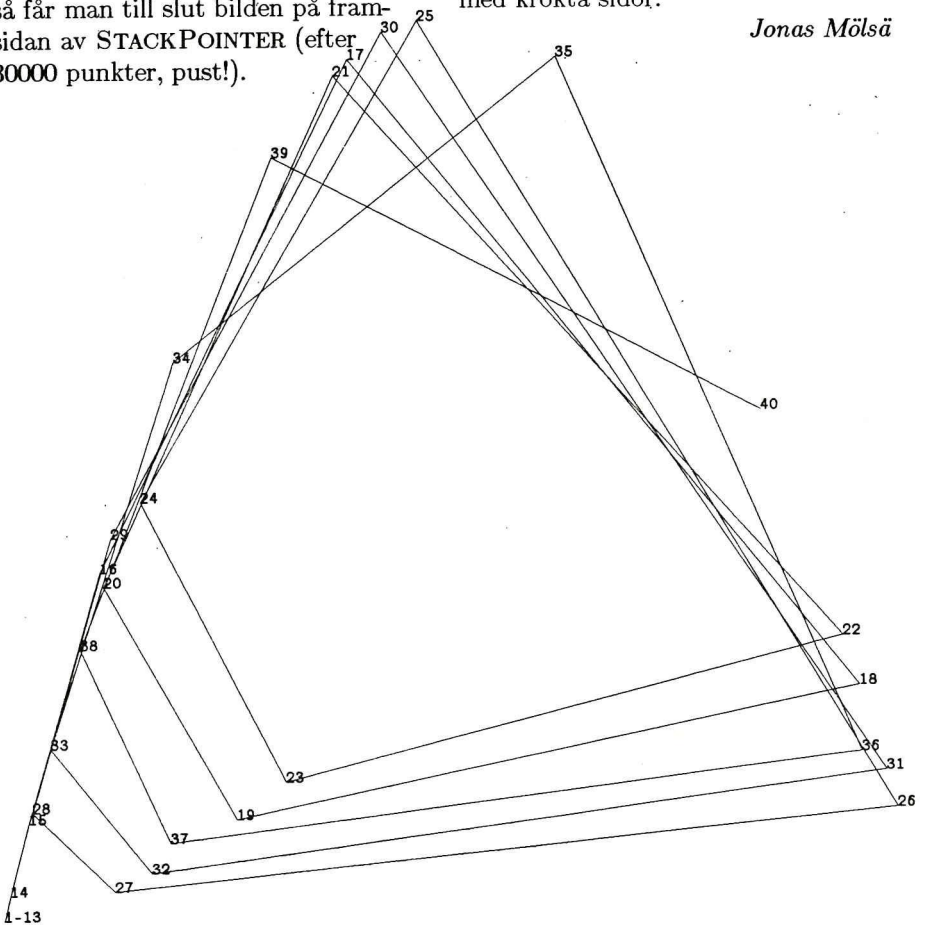
$y \leftarrow 4\lambda x(1-x)$
Plotta punkten (x, y)
$\lambda \leftarrow 1 - x^{1.85}$
$x \leftarrow y$

så får man till slut bilden på framsidan av STACKPOINTER (efter 30000 punkter, pust!).

Punkterna kommer huller om buller, men allihopa verkar ligga på olika kurvor. Här är de första 40 punkterna, hopbundna med linjer.

Om man byter ut 1.85 mot någonting annat så får man helt andra kurvor, t ex fyra spiraler eller en figur som nästan innehåller en triangel med krökta sidor.

Jonas Mölsä



```
BEGIN
  EXTERNAL CLASS elfin;
  elfin ("tty:",notext)
  BEGIN
    REAL x,y,lambda;
    graphics_on;
    scale(0,1,0,1);
    newpen(1);

    x:=lambda:=.00001;          ! Initiera X och Lambda ;

    WHILE TRUE DO BEGIN
      y:=4*lambda*x*(1-x);      ! Beräkna nytt funktionsvärde ;
      moveto(x,y);              ! Plotta                      ;
      drawto(x,y);
      lambda:=1-x**1.85;        ! Nya värden                  ;
      x:=y;                     !   till Lambda och X         ;
    END;

    graphics_off;
  END;
END;
```

Redaktörn

För några veckor sedan blev det anledning att titta på en dator med plasmaskärm. Intresset var inriktat mot läslighet, flimmer, efterlysning, m.m. Alltså glider jag in på närmaste Ericsson City. Blir mottagen av en dam som snabbt övertygas om att det inte är en skrivmaskin jag vill titta på. Fram skickas en av dessa nya försäljartalanger som kan sälja vad som helst (och gör det också). Jag blir informerad att jag gör ett klokt val. Det finns ju 'dos' på den. Det fanns också en massa 'ram', en 'ergodisk', den var hopfällbar och hur många program som helst. Jag påpekar lite försiktigt att jag var intresserad av att se datorn snurrande. Efter mer faktarabblande skulle jag få vänta en stund. Under tiden skulle en startdiskett hämtas. Disketten kom med försäljaren. Densamma instoppas i datorn. Datorn skriver att den är igång. Väntar så nu på att datum skall anges. Jag försöker knappa in. Ingenting händer. Jag försöker igen. Samma resultat. Försäljaren tittar på fodralet till disketten. Säger något om att 'det kanske inte är rätt diskett, men du kan ju se på det som är'. Jag skall alltså mha ca 25 tecken ojämnt fördelade på 3 rader försöka bilda mig en uppfattning om plasmaskärmen som arbetsredskap i jämförelse med traditionell skärm!

Jag läste att EIS gjorde 800 miljoner förlust under 1985. Dom kanske når 1 miljard under 1986. Heja Ericsson!

Ja, det är det hele.

/hn

Bjällran på katten

Katten var för ryslig. Alla råttorna var ense om att det inte kunde få fortsätta så här längre, och nu höll de rådslag om vad som var att göra.

De sammanträdde i en mörk, djup källare. Här kunde katten inte hitta dem, menade de. Och så började de diskutera.

Den ena sade ditt, och den andra sade datt, men det blev bara prat, och till sist sade en gammal klok råtta:

– Ni håller säkert med mig om att den största faran för oss är att vår fiende kommer smygande så tyst att vi inte hör honom. Om vi bara hörde när han kom, så skulle vi lättare kunna rädda oss. Jag föreslår därför att vi skaffar en liten bjällra och binder den om halsen på katten. På så sätt vet vi alltid om han är i närheten och kan skynda oss att gömma oss.

Råttorna tyckte det var en utmärkt idé och började dansa och hoppa, så glada var de att de inte längre behövde vara rädda för den rysliga katten.

Alla jublade av förtjusning ända tills en gammal råtta reste sig och sade:

– Allt det här är nog gott och väl. Men vem ska hänga bjällran om halsen på katten?

Råttorna såg på varandra. Ingen sade något. Det blev alldeles tyst i källaren, och än har inte katten fått bjällra om halsen.

Aisopos

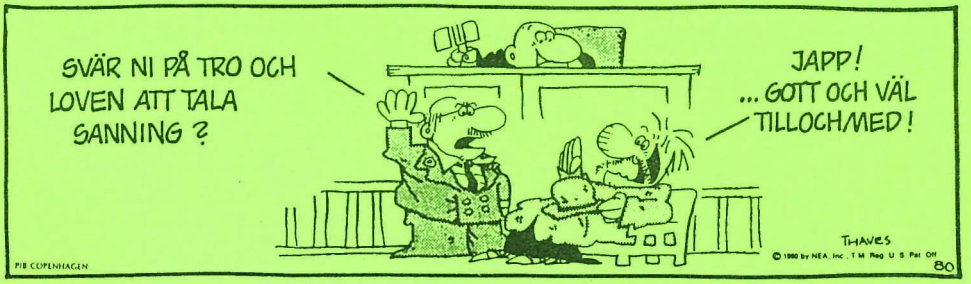
"Say 'cheese' "

Detta händer i Sollentuna C

En skön hack

```
******** PTYCON log,  2-Dec-85 21:24:01

******** 21:24:06 Connected to subjob (0)
TERMINAL TYPE VT100 PAGE 0 DEFER

.log 105,2002
JOB 16 Kicki Stockholm, 7.02 TTY75
ÄLGNJSP Other jobs same PPN:17Å
21:24    2-Dec-85       Mon

.sy z
 2    <OPR>     Operator      CTY 20_063 BACKUP 16+15   TI          2:04
14    10,626    M.Jansson      1   4_105 CC     19+35   TI            31
16    <SELF>    Sib           P11J17     SYSTAT 20+SPY  RN             1 $
17    <SELF>    Sib            3   4_101 PTYCON 24+34   SL            57
21    10,7257   D.Pettersson   4   4_103 DIRECT 10+24   ^C            31 $
24    10,302       , ' ,      P1J19      ROLEX  22+19   SL            43
25    10,302        ,   ,     P2J19      ROLEX  22+19   SL            43
26    10,302       , ,,,   ,  P3J19      ROLEX  20+19   SL            43
27    10,302        ,  ',  ,  P4J19      ROLEX  20+19   SL            43
28    10,302          ,  ;    P5J19      ROLEX  20+19   SL            44
29    10,302           '      P6J19      ROLEX  20+19   SL            44


.k

ÄLGTOUL Other users logged-in under Ä105,2002Å, Jobs: 17Å
Job 16  User SIB  Ä105,2002Å
Logged-off TTY75  at 21:24:44  on  2-Dec-85
Runtime: 0:00:01, KCS:28, Connect time: 0:00:27
Disk Reads:144, Writes:2
Session cost: 9 SEK, User factor: 1.0
```

Och rätt går den också! Ja, den där JMR ... */hn*



Bevingade ord av programmerare

1. Konstigt!
2. Det känner jag inte till!
3. Det har ju fungerat förr!
4. Det är bara några småsaker som ska fixas till!
5. Hur kunde det bli så?
6. Det måste vara ett hårdvarufel!
7. Dom kanske har bytt release?
8. Ni måste ha gjort något fel!
9. Ni måste ha något fel i indatan!
10. Men jag **har** inte ändrat något i den modulen!
11. Jodå, det måste bli klart innan dess!
12. Ni måste ha fått tag i en gammal version.
13. Det är bara ett skönhetsfel!
14. Jag är nästan klar!
15. Jodå, bara jag får införa de senaste ändringarna!
16. Det tar ingen tid!
17. Det beror antagligen på en olycklig slump.
18. Man **kan** inte testa allting!
19. Det **här** kan inte påverka det **där**!
20. Men jag trodde jag hade fixat det!
21. Det **finns** med, det är bara inte testat.
22. Egentligen fungerar det bra, fast det inte verkar så.
23. Frånsett att det inte funkar, hur tycker ni att det verkar?
24. Dokumentationen är på gång.
25. Nästan alla program är klara.
26. Programmet är i stort sett klart.
27. **Det** är inte mitt program!
28. Jag väntar på att de andra ska bli klara, så jag kan testa mitt program.
29. Det har varit ovanligt mycket strul!
30. Men förutsättningarna har ändrats hela tiden!
31. Jag **trodde** att jag hittat felet!
32. Det är bara en fem-minutersändring!
33. Jag ska **bara** ...