ETLA ELINKEINOELÄMÄN TUTKIMUSLAITOS
THE RESEARCH INSTITUTE OF THE FINNISH ECONOMY
Lönnrotinkatu 4 B, 00120 Helsinki 12, Finland, tel. 601322

Keskusteluaiheita
Discussion papers

Esko Torsti

MAT-OHJELMOINTITULKIN KÄYTTÖ
JA RAKENNE

No 290 11.05.1989

ISSN 0781-6847

This series consists of papers with limited circulation, intended to stimulate discussion. The papers must not be referred or quoted without the authors' permission.

TORSTI, Esko, MAT-OHJELMOINTITULKIN KÄYTTÖ JA RAKENNE, Helsinki : ETLA, Elinkeinoelämän Tutkimuslaitos, The Research Institute of the Finnish Economy, 1989. 67 s. (Keskusteluaiheita, Discussion Papers, ISSN 0781-6847 ; 290).

TIIVISTELMÄ: Keskustelualoitteessa esitellään mat-ohjelmointitulkin käyttöä ja rakennetta. Esitystä voi käyttää mat-käyttäjän käsikirjana, mutta myös ohjelman rakenteesta kiinnostunut saa karkean kuvauksen ohjelman toimintaperiaatteista. Ohjelma on ns. konetulkki, jonka toiminta perustuu välikoodin ja lohkottaisen tulkkauksen varaan. Ohjelma on rakennettu HPUX minikoneympäristöön, mutta se on periaatteessa helposti siirrettävissä esimerkiksi mikrotietokoneympäristöön.

ASIASANAT: C-ohjelmointi

TORSTI, Esko, USE AND STRUCTURE OF THE MAT PROGRAMMING INTERPRETER, Helsinki : ETLA, Elinkeinoelämän Tutkimuslaitos, The Research Institute of the Finnish Economy, 1989. 67 p. (Keskusteluaiheita, Discussion Papers, ISSN 0781-6847 ; 290).

ABSTRACT: This paper outlines the main features of the mat programming interpreter. It can also be used as a user manual. The program is a so-called machine interpreter, and the functioning of it is based on the use of intermediate code and structured interpreting. The program is implemented in an HPUX mini computer environment, but it can easily be moved to other computer environments, for example, in micro computers.

KEYWORDS: C-programming

SISÄLLYS

1. JOHDANTO 1

2. PERUSKÄSITTEITÄ 1

3. MUUTTUJAT 2
   3.1 MERKKIJONOMUUTTUJAT 3
   3.2 SKALAARIMUUTTUJAT 4
   3.3 VEKTORIT 5
   3.4 MATRIISIT 5

4. LAUSEET 6
   4.1 LUOKITTELU 6
   4.2 SIJOITUSLAUSE 7
   4.3 ALIOHJELMAN MÄÄRITTELY JA KUTSU 8
   4.4 RAKENTEISET LAUSEET 9
   4.5 LAUSEKKEET 12
   4.6 KOMMENTTILAUSEET 13

5. MATRIISIOPERAATIOT 13
   5.1 MATRIISIN KÄÄNTÄMINEN JA TRANSPONOINTI 13
   5.2 MATRIISIN ELEMENTTEIHIN KOHDISTUVAT OPERAATIOT 14
   5.3 RIVI- JA SARAKEOTSAKKEIDEN KÄSITTELY 15
   5.4 EPÄSUORAT SARAKEVIITTAUKSET 16
   5.5 MUUT MATRIISILAUSEET 17

6. TIEDOSTO-OPERAATIOT JA KÄYTTJÄRJESTELMÄKOMENNOT 20
   6.1 MUUTTUJIEN HAKU JA TALLETUS 20
   6.2 SPOOL-TIEDOSTOT 22
   6.3 AJOVIRTATIEDOSTOT JA RUN-KOMENTO 22
   6.4 UNIX:-KOMENTO 23

7. MUUT KÄSKYT 24
   7.1 PRINT 24
   7.2 INPUT JA READ 25
   7.3 SYMBOLS 26
   7.4 STATUS 26
   7.5 ALKUASETUKSET OHJELMAA KÄYNNISTETTÄESSÄ 27
   7.6 TULOSTUSTARKKUUDEN SÄÄTÄMINEN 27

8. REGRESSIOANALYYSI 27

9. VIRHEILMOITUKSET 29

10. KUVIOIDEN PIIRTÄMINEN 33

11. OHJELMAESIMERKKEJÄ 34
   11.1 REGRESSIOANALYYSI MATRIISIOPERAATIOILLA 34

11.2 PANOS-TUOTOS -MALLI 38

12. OHJELMAN TOIMINTAPERIAATTEISTA 41
12.1 OHJELMOINTITYÖKALUISTA 41
12.2 MAT-TULKIN SYNTAKSI JA KOODIN SUORITUS 44
12.3 MATIN TIETORAKENTEET 52
12.4 KESKEISIÄ FUNKTIOITA 55
12.5 SELAAJAN TOTEUTUS 61

LÄHTEET 67

1. JOHDANTO

Mat on C-kielellä ja Yacc-metakääntäjällä koodattu rakenteista ohjelmointia tukeva tulkkaava kieli. Mat-kieli muistuttaa osin esim. Pascalia ja C-kieltä, mutta syntaksi ei täysin vastaa mainittujen, tunnetumpien ohjelmointityökalujen syntakseja. Mat on toisaalta suppeampi, mutta toisaalta myös monipuolisempi väline kuin esikuvansa. Suppeampia piirteitä ovat mm. switch-lauseen ja osoitinmuuttujien puuttuminen. Monipuolistavia piirteitä ovat vektoreiden ja matriisien helppo käsittely, esim. keskeiset matriisioperaatiot ovat valmiina Matin syntaksissa. Ohjelman alkulähteet löytyvät kirjasta Kernighan-Pike: Unix programming environment. Kirjasta saatua mallia on edelleen laajennettu.

Ohjelma on tehty suurelta osin yhteistyönä Heikki Vajanteen kanssa, joka on kehittänyt ohjelman peruskehikkoa taloudellisten mallien suuntaan, kun taas tässä versiossa on pyritty painottamaan matriisilaskennan kannalta keskeisiä piirteitä. Heikki Vajanteen panos on ollut suuri myös regressiolohkon suunnittelussa. Mat-sovelluksia on rakentanut Eija Kauppi, joka on antanut Matista useita arvokkaita kehitysideoita ja mielipiteitä.

Mat-käsikirja on koottu siten, että alkuosassa käydään läpi Matin käyttöä ja lausesyntakseja. Loppuosassa esitellään muutamaa ohjelmaesimerkkiä ja hahmotellaan Mat-ohjelman koodausperiaatteita ja tärkeimpiä ohjelmalohkoja.

2. PERUSKÄSITTEITÄ

Perusperiaatteena Matissa on, että käyttäjä kirjoittaa Mat-syntaksin mukaista koodia tiedostoon tai suoraan standard inputiin, josta ohjelma lukee koodin ja suorittaa käyttäjän määrittelemät toiminnot. Mikäli käyttäjän antama ohjelmalause tai lauseryhmä ei ole Mat-syntaksin mukainen, ohjelma antaa ilmoituksen "syntax error". Toinen keskeinen virhetyyppi Matissa on "fatal error", jonka esiintyminen tarkoittaa muistialueylivuotoa. Tällainen ylivuoto johtuu yleensä siitä, että ohjelma ei ole pystynyt tekemään riittäviä dimensiotarkistuksia käyttäjän määrittelemille muuttujille. Todennäköisesti ohjelma toipuu täysin ylivuodosta, mutta testiajoissa on kohdattu muutamia tilanteita, joissa toipuminen ei ole ollut täydellistä. Matissa on myös useita tilannekohtaisia virheilmoituksia, joiden tarkoituksena on helpottaa lähinnä loogisten virheiden etsimistä.

3. MUUTTUJAT

Matissa numeeristen muuttujien tyyppinä on kaksoistarkkuuden liukulukuesitys, joka takaa parhaan mahdollisen laskentatarkkuuden. Mitään kokonaislukutyyppejä ei Matissa ole, vaan ohjelmointikielille tyypilliset kokonaislukutehtävät hoidetaan liukulukumuuttujilla. Skalaarimuuttujat allokoituvat automaattisesti symbolitauluun, kun taas matriisit ja vektorit on oletusarvotilassa kohdennettava muistiavaruuteen nimenomaisesti esimerkiksi dim-lauseella. Automaattiallokaatio tarkoittaa sitä, että muuttujia ei tarvitse julistaa, vaan muuttuja luodaan, kun sitä ensimmäisen kerran kutsutaan. Esim. tilanteessa "a = 3 + 4" muuttujalle "a" varataan tila, mikäli sitä aikaisemmin ei ole tehty. Jos "a" olisi matriisi, lausetta a[1,1] = 3 + 4" ei voida suorittaa ennen kuin matriisi julistetaan dim-lauseella. Mikäli matriisi luetaan suoraan tiedostosta, julistusta ei tarvitse tehdä. Symbolitaulun muuttujanimet ovat yksikäsitteisiä, ja tästä syystä matriisilla ja skalaarimuuttujalla ei voi olla samaa nimeä. Esimerkiksi useimmat Basic-tulkit pitävät eri tyyppisten muuttujien symbolitaulut erillään, ja tämä mahdollistaa saman tunnuksen käytön erityyppisille muuttujille. Tällaista ratkaisua ei ole Matissa kuitenkaan tehty, koska Mat sallii pitkien muuttujatunnusten (16 merkkiä) käytön. Käyttäjä ei siis käytännössä ikinä joudu pulaan muuttujanimimahdollisuuksien loppumisen kanssa.

Muuttujatunnus ei saa koostua mistä tahansa merkeistä, vaan laillisia merkkejä ovat kirjaimet, numerot ja alaviiva '_'. Muuttujatunnuksen ensimmäisen merkin on oltava kirjain. Lisäksi merkkijonomuuttujat sisältävät dollarimerkin, joka sijaitsee tunnuksen nimiosan lopussa ennen dimensiomääreitä.

Matissa on useita varattuja sanoja, joita ei voi käyttää muuttujaniminä. Näillä sanoilla on omat erityismerkityksensä käyttäjän antamaa koodia tulkattaessa. Varatut sanat ovat:

dim	rowsum	line
proc	to	bar
func	on	type
print	off	rem
print	span	pred
:	spool	precision
unix:	run	Graph
read	normi1	kill
input	normi2	join
modulo	normi3	string
return	names	turn
break	name	ux:
if	cmd	split
else	bread	status
while	bsave	colsum
for	with	res
diag	ind	reg

fread	dep	size
symbols	rem	autodim
row	change	col
fsave	canon	draw
pivot		

3.1 MERKKIJONOMUUTTUJAT

Merkkijonomuuttujan arvo koostuu käyttäjän haluamasta määrästä ASCII-merkkejä, eli kirjaimista, numeroista jne. Merkkijonomuuttujan arvo ilmaistaan perinteiseen tapaan lainausmerkkien välissä. Esim. "hei hei" on merkkijono, jonka pituus on seitsemän merkkiä (tai täsmällisesti ottaen pituus on kahdeksan merkkiä, kahdeksas merkki on loppumerkki). Merkkijonomuuttujia voi myös julistaa useampia saman nimen alle, jolloin muistiavaruuteen kohdennetaan merkkimatriisi, jonka jokaisella rivillä on yksi merkkijono (merkkijono on oikeastaan merkkivektori).

Merkkijonon julistus käy käskyllä

```
dim merkkijono $ 20
```

jossa dim on julistuskomento, merkkijono on julistettavan olion tunnus (nimi), '$'-merkki ilmaisee olion tyypin (=merkkijono) ja 20 kertoo merkkijonon pituuden.

Käskyllä

```
  dim merkkijono2 $ [ 20 ] 30
```

julistetaan 20 merkkijonoa saman symbolinimen taakse, kaikkien merkkijonojen pituus ollessa 30 merkkiä. Saman symbolin eri elementteihin viitataan indeksinumeroilla: Esim. taulukon kolmanteen merkkijonoon viitataan ilmauksella.

```
  merkkijono2 $ [ 2 ]
```

Kuten äskeisestä käy ilmi, kolmannen elementin indeksinumero on 2. Tämä johtuu siitä, että indeksointi Matissa noudattaa samaa logiikkaa kuin C-kielessä, ja näin ollen ensimmäisen elementin indeksinumero on 0. Numerointi alkaa siis nollasta.

Merkkijonomuuttujiin voidaan sijoittaa merkkijonoja sijoituksella:

```
merkkijono$ = "Hello, world"
```

Myös merkkijonomuuttujia voidaan sijoittaa toiseen merkkijonomuuttujaan:

```
 merkkijono2 $ [ 0 ] = merkkijono$
```

On muistettava, että dollarimerkin eli tyyppierottimen on aina oltava mukana tunnuksen yhteydessä, kun taas julistuksessa käytettävää kenttäpituusmäärettä ei enää julistuksen jälkeen saa käyttää.

3.2 SKALAARIMUUTTUJAT

Skalaarimuuttujien käyttö on yksinkertaista, sillä niitä käytettäessä ei ennakkojulistusta tarvitse tehdä. Tällaiseen muuttujaan voidaan suoraan sijoittaa jokin arvo:

```
Esim.   AABBCC = 3 + 4
```

Merkkijonotunnuksia voi käyttää päällekkäin numeeristen tunnusten kanssa, sillä symbolitaulussa '$' - merkki toimii yksikäsitteisenä erottimena. Tämä merkitsee luonnollisesti myös sitä, että numeerinen tunnus ei saa sisältää '$' - merkkiä.

```
Esim.  dim m_jono$20
       m_jono  = 999
       m_jono$ = "Hello, world"
```

on täysin laillinen ilmaisukokoelma, sillä '$' erottaa tunnukset yksikäsitteisesti toisistaan.

Eri asia on tietenkin se, miten järkevää on käyttää muuttujatunnuksia, jotka ovat hyvin lähellä toisiaan. Luultavasti on järkevää luoda ilmeikkäät ja helposti eroteltavat nimet eri muuttujille niiden tyypistä riippumatta.

Skalaarimuuttujien kohdentumisautomatiikka saattaa tietyissä tilanteissa aiheuttaa myös kiusallisia tilanteita, sillä esimerkiksi tilanteessa, jossa käyttäjä kirjoittaa käskyn "dim" vaikkapa muodossa "dam", symbolitauluun ilmestyy skalaarimuuttuja "dam", vaikka sitä ei tarvittaisikaan. Yhden skalaarimuuttujan tilanvaraus on hyvin marginaalinen ohjelman tehokkuuden kannalta, mutta mikäli nimelle "dam" myöhemmin saman ohjelmaistunnon aikana haluttaisiin kohdentaa esim. matriisi, tämä ei onnistu, koska tunnus on jo varattu.

Symbolitaulun elementti voidaan tuhota kill-komennolla. Tällöin symboli poistetaan taulusta, ja uusi elementti voidaan kohdentaa samalle tunnukselle. Kill-komento ei kuitenkaan aidosti vapauta varattua tilaa, vaan ainoastaan vallitseva mat-prosessi pystyy käyttämään vapautettua tilaa. Jos haluaa tuhota esimerkiksi isoja matriiseja, on usein järkevämpää tallettaa tarpeelliset symbolitauluelementit ja aloittaa uusi mat-prosessi. Tällainen menettely on edullista koko koneen hyödyntämisen eli käyttäjien yhteisen edun kannalta.

Mikäli käsittelee matriisia, jonka sarakkeet on varustettu nimillä, mat-

riisin sarakemuuttujaan elementtiin voidaan viitata myös seuraavasti:

< matriisi >.< sarakenimi > [< indeksi >]

Esimerkki: On määritelty matriisi "koe", jonka ensimmäisellä sarakkella on otsake "ensi". Tällöin matriisin yläkulman alkioon voidaan viitata seuraavasti:

koe.ensi [0]

Tämä on täysin vastaava kuin esitys

koe [0][0]

Tämän vaihtoehtoisen skalaariviittauksn tarkoituksena on helpottaa muuttujien käsittelyä. Otsakeviittauksen varjopuolena on hieman raskaampi rakenne ja täten myös hieman pitempi suoritusaika. Mitään suurta nopeuseroa ei näillä vaihtoehtoisilla tavoilla kuitenkaan ole.

3.3 **VEKTORIT**

Vektoreita voidaan julistaa paitsi merkkijonoille myös numeerisille elementeille. Julistus tapahtuu dim-lauseella:

```
dim vektori1 [ 30 ], vektori2 [ 50 ]
```

Vektorin eri elementteihin viitataan nollasta alkavilla indeksinumeroilla (0 <= indeksi < vektorin koko).

Vektoreiden käyttöaluetta rajoittaa se, että useat Matin operaatiot on määritelty ainoastaan matriiseille, ja tästä syystä vektorien asemesta on usein käytettävä n x 1 tai 1 x n matriiseja.

3.4 **MATRIISIT**

Matriisit julistetaan dim-lauseella, esim:

```
dim matri1 [ 30,30 ], pikku [ 5,1 ]
```

Elementteihin viitataan nollasta alkavilla indeksinumeroilla.

Kuten aikaisemmin jo todettiin, useissa vektorityyppisissäkin operaatioissa joudutaan turvautumaan 1 x n tai n x 1 matriiseihin, jotka tulkinnallisesti ovat vektoreita. N x 1-matriisin ja "aidon" vektorin erona on erilainen tietorakenne, ja näitä kahta tulkinnallisesti samantyyppistä oliota erottaa myös Matin jäsentäjän symbolityyppi. Mikäli olio on

julistettu matriisiksi, jäsentäjä kohtelee sitä eri tavalla kuin jos olio olisi julistettu vektoriksi (Jäsentäjä on se Matin ohjelmalohko, joka tutkii syötettyjen lauseiden syntaktisen järkevyyden eli sen, että syöte on määritellyn kieliopin mukainen). Tietorakenteeltaan aito vektori on hieman kevyempi, mutta sen käyttöalue on huomattavasti suppeampi.

4. LAUSEET

Matissa käyttäjän antamat käskyt koostuvat lauseista. Lauseet erotetaan toisistaan EOST-merkillä (end of statement), joka on määritelty rivinvaihdoksi '\n' tai puolipisteeksi ';'. Mat ei välttämättä suorita lausetta heti EOST-merkin kohdattuaan, vaan se saattaa koota monia lauseita pinoon ja suorittaa niitä useamman kerrallaan. Tälläisella järjestelyllä saavutetaan se etu, että samat lauset voidaan suorittaa useamman kerran vain yhdellä jäsennyksellä, ja tietyissä operaatioissa tällä saavutetetaan huomattavasti nopeampia suoritusaikoja.

Mikäli lauseen haluaa sijoittaa useammalle riville, rivinvaihtomerkin EOST-rooli voidaan ohittaa kirjoittamalla rivin loppuun merkki '@' . Näin kuvaruudun leveys ei aseta rajoitteita kirjoitettavien lausekkeiden pituudelle.

Esimerkki:

```
a = 12321321 + 123123213 / 12342111 * 3456   @
  - 3.1415927
```

Rivin lopussa oleva kissanhäntä kertoo jäsentäjälle, että lause jatkuu vielä seuraavalla rivillä.

Luvussa 4 käsitellään skalaari- tai merkkijonolauseita. Vaikka matriiseillekin on määritelty lauseita, niitä tarkastellaan vasta luvussa 5.

4.1 LUOKITTELU

Lauseet voidaan jakaa yksinkertaisiin ja rakenteisiin lauseisiin sen mukaan, onko lause osana toista lausetta vai ei. Yksinkertaisia lauseita ovat sijoituslause ja aliohjelmakutsu, kun taas rakenteisia lauseita ovat valinta- ja toistolauseet.

4.2 SIJOITUSLAUSE

Sijoituslause (assignment statement) on yleisimpiä koodauksessa käytettäviä lauseita. Sen tarkoituksena on siirtää jonkin lausekkeen arvo johonkin muuttujaan. Lauseke taas voi olla toinen muuttuja, aritmeettinen lauseke tai näiden kombinaatio. Luvussa 3 mainittiin esimerkkejä sijoituslauseen käytöstä. Sijoitettavien olioiden on oltava tyyppiyhtennsopivia, ja täten esim. merkkijomuuttujaan ei voi sijoittaa numeerista matriisia.

Sijoituslauseen syntaksi on:

<muuttuja> '=' <lauseke> , jossa

<muuttuja>: skalaari | merkkijono

<lauseke>: aritm. lauseke | merkkijonolauseke

Vaikka sijoituslause sisältääkin '='- merkin, tämä ei tarkoita väittämää siitä, että jokin asia on jotakin. Sijoituslauseella on aina suunta oikealta vasemmalle, ja lause on tulkittava siten, että jokin (vasemmalla sijaitseva) olio saa arvokseen oikealla sijaitsevan tyyppiyhteeensopivan olion arvon. '='-merkin oikea verbaalinen vastine tässä yhteydessä on osoitinmerkki.

Sijoituslausessa voi '=' - merkin asemesta käyttää merkkejä '+=', '+-', '+*' tai '+/'. Esimerkiksi merkki '+=' tarkoittaa sitä, että sijoituslauseen vasemmalla puolella olevan muuttujan sisältöön lisätään oikealla puolella olevan muuttujan tai lausekkeen arvo. Vastaavasti '/=' tarkoittaa sitä, että lauseen vasemmalla puolella sijaitsevan muuttujan sisältö jaetaan oikealla puolella sijaitsevan muuttujan tai lausekkeen arvolla.

Sijoituslauseessa on myös mahdollista laskea n x 1 matriisista ensimmäinen, toinen ja kolmas vektorinormi skalaarimuuttujaan. Syntaksi on tällöin:

<skalaarimuuttuja> = norm1 <matriisi>
<skalaarimuuttuja> = norm2 <matriisi>
<skalaarimuuttuja> = norm3 <matriisi>

Eräs sijoituslausetyyppi on myös numeerisen skalaarimuuttujan arvon lisääminen tai vähentäminen yhdellä.

lisäys: <skalaarimuuttuja>++
vähennys: <skalaarimuuttuja>--

Esimerkki: Kaksi vaihtoehtoista tapaa muuttujan i arvon kasvattamiseksi

yhdellä:

i++ on sama kuin i = i + 1.

4.3 ALIOHJELMAN MÄÄRITTELY JA KUTSU

Matissa on kahdentyyppisiä aliohjelmia: proseduureja ja funktioita. Näiden kahden aliohjelmatyypin erona on se, että funktio palauttaa skalaariarvon mutta proseduuri ei.

4.3.1 PROSEDUURIN MÄÄRITTELY

```
proc <procname> ( ) {

      ... suoritettavat lauseet ....

  }
```

Proseduurin kutsu:

```
<procname> ( )
```

Esim: Proseduuri, joka kaiuttaa päätteelle merkkijonoklassikon:

```
proc hello() {
    print "Hello, world\n"
  }
```

Ja kutsu:

```
hello()
```

Aliohjelmille voidaan antaa parametreina skalaarimuuttujia. Annettuihin parametreihin viitataan proseduurin sisällä merkinnällä %n, n tarkoittaa parametrin järjestysnumeroa.

Esimerkki: Proseduuri, joka kaiuttaa päätteelle suurimman kahdesta annetusta parametrista (mikäli parametrit ovat erisuuria)

```
proc koe2( ) {
   if ( %1 < %2 ) print %2  else  print %1
 }
```

Kutsu:

```
koe2( 3,4 )
```

Mikäli aliohjelman sisällä käytetään parametreja, on niitä käytettävä myös aliohjelmakutsussa tai kutsun suoritus päättyy virheilmoitukseen.

4.3.2 FUNKTION MÄÄRITTELY Funktio palauttaa numeerisen skalaarimuuttujan, muutoin funktion käyttö vastaa täysin proseduurien käyttöä.

Esimerkki:

```
func koefunktio( ) {
    if ( %1 > 0 )  a = %1
    return a
  }
```

Kutsu:

```
c = koefunktio( 9 )
```

4.4 RAKENTEISET LAUSEET

Rakenteisia lauseita ovat valinta- ja toistolauseet. Valintalauseita taas ovat if-else -lauseet ja toistolauseita ovat while- ja for- lauseet.

Rakenteiset lauseet ovat ehdollisia käyttäjän määrittelemälle ehdolle. Ehto sijoitetaan sulkumerkkien sisään rakenteisen lauseen tunnusosan (for / while / if) perään.

<ehto> : ehto on looginen totuuslause, joka palauttaa skalaarimuuttujan. Jos palautettava arvo on nolla, on looginen lause epätosi, ja nollasta poikkeavat arvot ovat tosia. Tätä ominaisuutta voi joskus käyttää hyväkseen, esimerkiksi lause " if (1) " on aina totta.

Varsinaiset ehdoissa käytettävät loogiset operaatiot ovat (a ja b numeerisia skalaarimuuttujia):

a	>	b	suurempi kuin
a	<	b	pienempi kuin
a	>=	b	suurempi ja yhtäsuuri kuin
a	<=	b	pienempi ja yhtäsuuri kuin
a	==	b	yhtäsuuri kuin

```
a  !=   b              erisuuri kuin
a  &&   b              a ja b  (konjunktio)
a  ||   b              a tai b (disjunktio)
!a                     ei a (negaatio)
```

Myös merkkijonomuuttujia voidaan verrata.
Merkkijonoilla yhtäsuuruuden testaus tarkoittaa sitä, että merkkijonoja verrataan niiden alusta lähtien merkkien ASCII-koodien mukaan. Erisuuruus syntyy, kun merkkijonosta löytyy ensimmäinen erilainen merkki.

Esim. "koira " ja "koiruus" ,

jossa erisuuruus syntyy viidensien merkkien "a" ja "u" kohdalla koiran ollessa suurempi kuin koiruuden.

Merkkijonojen vertailuoperaatiot
(a_jono ja b_jono merkkijonoja):

```
a_jono   >   b_jono
a_jono   >=  b_jono
a_jono   <   b_jono
a_jono   <=  b_jono
a_jono   ==  b_jono
a_jono   !=  b_jono
```

4.4.1 IF - ELSE

```
if (   <ehto>   ) {

 ... suoritettavat lauseet  ...

} else {

 ... suoritettavat lauseet ...

}
```

Else-osa ei ole pakollinen, vaan se voidaan jättää pois. if - lausetta voi myös käyttää yksinkertaisena lauseena (= ei-rakenteisena lauseena), jolloin lohkomerkkejä '{' ja '}' ei tarvitse käyttää. Tällöin ehto-osaa seuraa ennen rivinvaihtoa ehdon toteutuessa suoritettava lause. Myös mahdollisen else - osan on oltava samalla rivillä silloin, kun if - lauseen kanssa käytetään ei-rakenteisia lauseita. Mikäli suoritettavat lauseet haluaa sijoittaa eri riveille kuin ehtolauseen nimiosan, rivisidonnaisuudesta pääsee eroon kissanhäntämerkillä '@'.

4.4.2 WHILE While-lausetta suoritetaan niin kauan kuin sulkumerkkien sisällä määritelty ehto on tosi.

```
while (  <ehto>   )  {

  ... suoritettavat lauseet ...

}
```

Ehto-osa on samanlainen kuin if-lauseessakin. If:n ja while:n ero on siinä, että if suoritetaan vain kerran, mutta while suoritetaan toistuvasti niin kauan kun ehto on tosi.

Esimerkki: lukujen luettelu yhdestä sataan.

```
i=0
while ( i < 100 ) {
  print i, "\n"
  i = i + 1
 }
```

Print-lauseessa esiintyvä kummajainen "\n" on rivinvaihtomerkki, joten jokainen tulostettava luku sijoitetaan omalle rivilleen.

4.4.3 FOR For-lause on toistettava rakenteinen lause kuten while-lausekin. For-lauseen erona on kuitenkin se, että sen syntaksiin kuuluu paitsi ehto-osa myös ehtomuuttujan inkrementointi ja alkuarvon asetus.

Syntaksi:

```
for ( <ehtom> =  <alkua> , <ehtom> <loog op> <loppua> , <inkr> ) {

    ... suoritettavat lauseet .......

 }
```

jossa

<ehtom>	ehtomuuttuja
<alkua>	alkuarvo
<loog op>	looginen operaattori (<, >, ...)
<loppua>	loppuarvo
<inkr>	ehtomuuttujan lisäys

Esimerkki: Äskeinen esimerkki for-lauseella

```
for ( i=0,  i < 100, i++ ) {
  print i, "\n"
 }
```

Kuten esimerkistä voidaan huomata, for-lauseella päästään eräissä tapauksissa siistimpään ulkoasuun.

4.5 LAUSEKKEET

Lausekkeita voidaan käyttää paitsi sijoituslauseiden yhteydessä myös suoraan esim print-lauseen kanssa, jolloin lausekkeen arvoa ei talleteta minkään muuttujan arvoksi.

Esim. print 3+4+5*8

tulostaa lausekkeen arvon 47, mutta tulos ei tallennu minnekään.

Lausekkeet voivat olla mielivaltaisen pituisia (järkevissä rajoissa), ne voivat sisältää sulkuja, muuttujia, lukuja tai C-kirjastossa olevia matemaattisia funktioita. Mahdolliset funktiot ovat:

sin(x),
cos(x),
atan(x),
log(x),
log10(x),
exp(x),
sqrt(x) ja
abs(x).

Myös loogisia ehtoja voidaan käyttää lausekkeissa (ks. kohta 4.4), ne palauttavat arvon 1 väittämän ollessa tosi ja arvon 0, kun väittämä on epätosi.

Esim:

```
y  =  (ve + x - ve)                    @
    * ve                               @
    * ((x >= ve) && (x < ve))
```

Mikäli lauseen viimeisellä rivillä oleva väittämä on epätosi, y saa aina arvon nolla.

Lausekkeessa voi olla:

lausekkeita	
lukuja	(numerot, desimaalipiste, E, + ja -)
sulkulausekkeita	
skalaarimuuttujia	(vektorin elementti on skalaari)
funktioita	(C-kirjasto ja omat)
lauseke +,-,*,/ lauseke	(summa, erotus, tulo, osamäärä)
lauseke ^ lauseke	(potenssiinkorotus)
- lauseke	(unary - miinus)
loogisia lausekkeita	(>, < jne...)

4.6 KOMMENTTILAUSEET

Kommenttilauseet alkavat merkillä "/*" ja päättyvät merkkiin "*/". Kaikki näiden merkkien välissä oleva teksti tulkitaan kommentiksi.

5. MATRIISIOPERAATIOT

Matissa on sisäänrakennettuna joitakin matriisioperaatioita. Matriisilauseiden jäsentäjä ei salli kahta matriisia pitempiä ilmauksia. Täten esim. ilmaus "a = b + c + d" on kielletty. Tällaiseen ratkaisuun on päädytty, koska jäsennettäessä edellä mainittua lausetta tulos c+d on ensin talletettava väliaikaisesti jonnekin ennen kuin se voidaan lisätä b-matriisin ja sijoittaa matriisiin a. Koska matriisit saattavat pahimmillaan olla hyvinkin suuria, tällaisesta ylimääräisestä tilanvarauksesta saattaisi kasaantua huomattaviakin turhia tilanvarauksia. Näin ollen edellinen ongelma on ratkaistava kahdessa osassa:

```
a = b + c
a = a + d
```

5.1 MATRIISIN KÄÄNTÄMINEN JA TRANSPONOINTI

Matriisin käännetään kirjassa Wilkinson, Reinsch: Linear Algebra sivuilla 119-133 esitetyn algoritmin mukaisesti.

Lauseen syntaksi on:

<matriisi> = inv (<matriisi>)

Matriisien on oltava samankokoisia neliömatriiseja. Tulosmatriisi ei synny automaattisesti, vaan se on luotava ennen kääntölausetta esim. dim-lauseella (Toinen tapa matriisin luomiseksi ja tarvittavan tilan varaamiseksi on lukea matriisi tiedostosta).

Matriiseja voidaan dimensioida myös automaattisesti, mutta tällainen toiminto edellyttää autodim-lauseen käyttöä. Antamalla autodim-komento uuden muuttujan oletustyyppi on matriisi, muussa tapauksessa oletusarvo on skalaari. Kirjoittamalla uudestaan autodim, vaihdetaan uuden muuttujan oletusarvoksi jälleen skalaari.

Käännetty matriisi voidaan myös sijoittaa itseensä eli matriisi voidaan kääntää paikallaan. On muistettava, että matriisia ei voida transponoida paikallaan, vaan näin tehtäessä tulos on arvaamaton ja luultavasti väärä.

Matriisi transponoidaan seuraavasti:

<matriisi> = trn (<matriisi>)

Esim. a = inv(a) antaa järkevän tuloksen,
mutta a = trn(a) antaa väärän tuloksen

Matriisin kääntöalgoritmi ei toimi, mikäli liikutaan koneen laskutarkkuuden äärimmäisillä rajoilla. (esimerkiksi jos käännettävä matriisi on hyvin lähellä singulaarista matriisia, tulokset saattavat olla vääriä) Hyvin lähellä singulaarista tarkoittaa esim. matriisia:

1260	2628	6312
2628	5484	13176
6312	13176	31664

Matriisi on ensi silmäyksellä hyvin kiltin näköinen, mutta se paljastaa todellisen luonteensa kun jaetaan toisen sarakkeen alkiot ensimmäisen sarakkeen alkioilla. Tällöin paljastuu lähes täydellinen lineaarinen riippuvuus, josta kääntöalgoritmi ja laskutarkkuus eivät pysty selviytymään. Ongelma on tietenkin selvä, sillä likimain singulaarisella matriisilla ei käytännössä ole käänteismatriisia. Hankalaksi tilanteen tekee se, että kääntöalgoritmi ei tarkista sitä, onko kääntäminen suoritettu oikein vai ei.

Lähes singulaarisia matriiseja on harvassa, ja mikäli epäilyksiä syntyy, kääntöfunktion toiminta voidaan tarkistaa kertomalla matriisi käänteismatriisillaan ja toteamalla tuloksen olevan yksikkömatriisi. Mikäli matriisi on täysin singulaarinen, kääntölause päättyy virheilmoitukseen.

5.2 MATRIISIN ELEMENTTEIHIN KOHDISTUVAT OPERAATIOT

Matriisin jokainen alkio on numeerinen skalaarimuuttuja, jota voidaan käyttää aivan kuten aitoakin skalaaria. Viittaus matriisin alkioon on seuraava:

<matriisi> [<ind>][<ind>]

<ind> on lauseke. On muistettava, että matriisin indeksiviittaukset alkavat indeksistä 0 viimeisen indeksin ollessa matriisin rivi/sarakeulottuvuus - 1. Matriisin elementtiviittauksen ensimmäinen indeksi tarkoittaa riviä ja toinen saraketta.

5.3 RIVI- JA SARAKEOTSAKKEIDEN KÄSITTELY

Aina kun matriisi allokoidaan, sille varataan data-alueen lisäksi tilaa myös rivi- ja sarakeotsakkeille. Kun matriisi sijoitetaan toiseen matriisiin, myös nimet kopioidaan matriisista toiseen. Kun matriisi julistetaan dim-komennolla, matriisiin sijoitetaan oletusnimet. Rivit merkitään nollasta alkavilla järjestysnumeroilla ja sarakkeille annetaan COLx-tyyppiset nimet, jossa x viittaa sarakkeen järjestysnumeroon. Oletusnimet voidaan vaihtaan sopiviksi joko suoraan erityisillä otsakekomennoilla tai esimerkiksi sijoittamalla julistetun matriisin elementtien arvoksi jonkin toisen matriisin elementtien arvot. Tällöin myös nimet sijoitetaan matriisista toiseen. Sama koskee myös useita muita matriisisijoitusoperaatioita. Esimerkiksi kahden matriisin elementtien summaoperaatiossa ensimmäisen summandin nimet kopioidaan tulosmatriisiin.

Esimerkki:

```
Julistetaan uusi matriisi:
  dim uusi [ 20, 5 ]   (perusotsakkeilla)
Luetaan matriisi tiedostosta:
  fread "vanha" (sisältää nimet)
Sijoitetaan:
   uusi = vanha  (uudessa vanhan nimet)
```

Matriisin nimet voidaan tulostaa komennolla:

name print <matriisi>

Mikäli nimiä ei matriisissa ole, tulostuskomennon yhteydessä annetaan virheilmoitus. Käytännössä ei juuri koskaan esiinny tilanteita, joissa matriisiotsakkeita ei ole olemassa, joten tätä virheilmoitusta ei käyttäjä todennäköisesti näe.

Matriisinimien maksimipituus on 16 merkkiä, mutta pitkät nimet saattavat aikaansaada ongelmia mm. epäsuorien sarakeviittauksien yhteydessä. Selaajaan on määritelty yksittäisen symbolielementin nimen maksimipituus, ja koska epäsuora sarakeviittaus on oma symbolinsa, sen yhteispituuden maksimi (pituus on matriisinimen, sarakenimen ja liitospisteen merkkien lukumäärän summa) saattaa helposti ylittyä. Joskus on kuitenkin tärkeää saada käyttää pitkiä nimiä, ja tästä syystä maksimipituudesta huolehtiminen on epäsuorien sarakeviittauksien osalta jätetty käyttäjän huoleksi. Mikäli epäsuorat symbolitunukset ovat liian pitkiä, tästä annetaan virheilmoitus.

```
/* Tiedosto /users/int/et/teksti/dokut/demo4  ET 26.8.88  */

/* Tunnusylivuodot epäsuorien sarakeviittauksien yhteydessä  */

spool on "tulos2"
dim matriisi [ 10,10 ]             /*  Julistetaan matriisi   */
matriisi 3 col name = "heiheihei" /*  Muutetaan oletusnimiä  */
matriisi.COL0 = 35.8              /*  Epäsuora viittaus      */
matriisi.heiheihei = 7.2          /*  Mutta...               */
spool off
```

Tulostiedosto tulos2 näyttää seuraavalta:

```
dim  matriisi [ 10,10 ]             /*/  Julistetaan matriisi  */
matriisi  3 col name = "heiheihei" /*/  Muutetaan oletusnimiä */
matriisi.COL0  =  35.8             /*/  Epäsuoria viittaus    */
mat: name too long matriisi.heiheihi = 7.2 /* Mutta... */
spool  off
```

Virheilmoitus on tulostiedostossa siinä kohdassa, jossa jäsentäjä huomasi epäsopivan ilmauksen. Lause 'matriisi.COL0 = 35.8' onnistui, koska nimen kokonaispituus ei ollut selaajan mielestä liian pitkä. Tunnus 'matriisi.heiheihei' sen sijaan oli liian pitkä, joten suoritus pysähtyi siihen. Kun suoritus päättyy virheeseen, tiedoston loppuosa vyörytetään selaajan läpi ilman jäsennystä ja suoritusta. Mikäli jonossa on seuraava tiedosto, kone puhdistetaan ja suoritusta jatketaan siitä. Muussa tapauksessa suoritus lopetetaan. Yleensä seuraava tiedosto on standard input, joten ohjelman kontrolli palaa käyttäjän päätteelle.

Matriisinimiä voidaan vaihtaa vaihtaa komennolla:

```
<matriisi> <lauseke> row name = <merkkijonomuuttuja>
<matriisi> <lauseke> col name = <merkkijonomuuttuja>
```

Lauseke voi olla mikä tahansa skalaarilauseke ja merkkijonomuuttujan asemesta voi käyttää myös merkkijonoa.

Esim:

```
  a_matriisi 3+4-2 row name = "Hello, world"
```

5.4 EPÄSUORAT SARAKEVIITTAUKSET

Kuten kohdassa 3.2 jo todettiin, matriisin sarakkeeseen voidaan viitata epäsuorasti sen sarakenimen avulla. Kohdassa 3.2 mainittu viittaus oli muotoa

```
koe.ensi [ 0 ],
```

jolla viitattiin matriisin koe ensi-nimisen sarakkeen ensimmäiseen alkioon. Jättämällä hakasulkuosa pois lausekkeesta voidaan viitata koko sarakkeeseen eli esimerkiksi kokonaiseen aikasarjaan. Jos kuvataan epäsuoraa viittausta <matriisi>.< sarakenimi > (esimerkiksi koe.ensi) merkinnällä <savi> (=sarakeviittaus), sarakeviittauksille mahdolliset syntaktiset rakenteet voidaan ilmoittaa seuraavasti:

```
<savi> = <lauseke>,                 sijoitus
<savi> = <savi> + <savi>,           summa
<savi> = <savi> - <savi>,           erotus
<savi> = <savi> * <savi>,           tulo
<savi> = <savi> / <savi>,           osamäärä
<savi> = <matriisi>,                matriisin (1. sarakkeen) sijoitus
<matriisi> = <savi>,                sarake matriisin (1.) sarakkeeseen
<savi> = <matriisi> <lauseke> col, matriisin sarakkeen sijoittaminen
<savi> = <matriisi> <lauseke> row, matriisin rivin sijoittaminen
<savi> = <savi>,                    sarake toiseen sarakkeeseen
change <savi> , <savi>,           kahden sarakkeen vaihto
<savi> = <savi> + <lauseke>,        skalaarin ja sarakkeen väl. oper.
<savi> = <savi> - <lauseke>                    .
<savi> = <savi> * <lauseke>                    .
<savi> = <savi> / <lauseke>                    .
<savi> = <lauseke> + <savi>                    .
<savi> = <lauseke> - <savi>                    .
<savi> = <lauseke> * <savi>                    .
<savi> = <lauseke> / <savi>
```

Sarakeviittausten yhteydessä on mahdollista käyttää lag-, lead-, rel- ja logdif- funktioita. Lag viivästää matriisin sarakkeella olevaa havaintovektoria ja lead vastaavasti edistää havaintovektoria. Rel laskee prosenttimuutoksen ja logdif logaritmisen muutoksen matriisin sarakkeesta elementeittäin. Syntaksit ovat:

```
<savi> = logdif( <savi>, <lauseke> )
<savi> = lag(    <savi>, <lauseke> )
<savi> = lead(   <savi>, <lauseke> )
<savi> = rel(    <savi>, <lauseke> )
```

Esimerkki:

```
koe.ensi =  logdif( koe.toinen, 2 )
```

5.5 MUUT MATRIISILAUSEET

Matriisilauseiden koot on tilan säästämiseksi rajoitettu sellaisiksi, että apumuuttujia ei evaluointivaiheessa tarvita. Matriisien täytyy olla periaatteessa kokoyhteensopivia, mutta matriisin sijoitus toiseen mat-

riisin onnistuu, vaikka matriisit olisivatkin erikokoisia. Tällöin matriisista sijoitetaan vasemmasta yläkulmasta niin paljon kuin pienemmän matriisin koko sallii. Tällaisessa erisuuruustapauksessa sijoitus siis suoritetaan, mutta Mat antaa varoituksen havaitusta erisuuruudesta.

Käytössä olevat matriisilauseet ovat:

```
<matriisi> = <matriisi> + <matriisi> ,     elementtien summaus
<matriisi> = <matriisi> - <matriisi> ,     elementtien erotus
<matriisi> = <matriisi> * <matriisi> ,     matriisitulo
<matriisi> = <matriisi> / <matriisi> ,     elementtien osamäärä
<matriisi> = <matriisi> ? <matriisi> ,     elementtien tulo
<matriisi> = <matriisi> ,                  matriisin toiseen matriisiin
name print <matriisi> ,                    matriisin otsakkeiden tulostus
print <matriisi> ,                         matriisin tulostus
row <lauseke> to <lauseke> ,               rivitulostusjänteen valinta
col <lauseke> to <lauseke> ,               saraketulostusjänteen valinta
<matriisi> = diag <lauseke> ,              diagonaalielementin sijoitus
<matriisi> = mean( <matriisi> ) ,          keskiarvomatriisi
<matriisi> = dev( <matriisi> ) ,           hajontamatriisi
<matriisi> = cov( <matriisi> ) ,           kovarianssimatriisi
<matriisi> = corr( <matriisi> ) ,          korrelaatiomatriisi
join <matriisi>,<matriisi> to <matriisi>, matriisien liittäminen
turn <matriisi>,                           rivien kääntäminen

<matriisi> = pivot <matriisi> , <lauseke>,              pivot-operaatio
change <matriisi> <lauseke> col, <matriisi> <lauseke> col, sarakkeiden vaihto
change <matriisi> <lauseke> row, <matriisi> <lauseke> row, rivien vaihto
split <matriisi> to <merkkijono> , <merkkijono>,        *)
```

*) Split-komennolla matriisi voidaan jakaa varsinaiseen aineistoon ja otsakkeisiin. Jos matriisi halutaan esimerkiksi siirtää toiseen ohjelmaympäristöön, joka pystyy lukemaa ainoastaa numeerista aineistoa, erottelu voidaan tehdä split-komennolla. Komennossa käytettävät merkkijonot tarkoittavat niitä kahta tiedostoa, joihin aineisto ja otsakkeet tulostuvat. Käsky on kätevä esimerkiksi silloin, kun HP9000-ympäristöstä siirretään aineistoja mikrotietokoneen PC-GIVE -ohjelmistoon.

Joskus matriisin jokaiseen alkioon halutaan esimerkiksi lisätä tai vähentää jokin skalaari ja tulos halutaan sijoittaa johonkin toiseen matriisin. Seuraavat syntaksit ovat tällaisia tilanteita varten:

```
<matriisi> = <matriisi> + <lauseke>
<matriisi> = <matriisi> - <lauseke>
<matriisi> = <matriisi> * <lauseke>
<matriisi> = <matriisi> / <lauseke>
<matriisi> = <lauseke>  + <matriisi>
<matriisi> = <lauseke>  - <matriisi>
<matriisi> = <lauseke>  * <matriisi>
<matriisi> = <lauseke>  / <matriisi>
```

Matriisin jonkin rivin tai sarakkeen sijoittaminen toisen matriisin ensimmäiseksi riviksi tai sarakkeeksi:

<matriisi> = <matriisi> <lauseke> row
<matriisi> = <matriisi> <lauseke> col

Matriisin jonkin rivin tai sarakkeen sijoittaminen toisen matriisin joksikin riviksi tai sarakkeeksi:

<matriisi> <lauseke> col = <matriisi> <lauseke> col
<matriisi> <lauseke> row = <matriisi> <lauseke> row

Luvun sijoittaminen matriisin jokaiseen alkioon:

<matriisi> = <lauseke>

Rivi- ja sarakesumman laskeminen (tulosmatriisit ovat n x 1 tai 1 x n matriiseja).

<matriisi> = colsum <matriisi>
<matriisi> = rowsum <matriisi>

size <matriisi> , matriisin koko

Mikäli jonkin matriisioperaation tulosmatriisin dimensiota ei haluta määritellä dim-lauseella, tulosmatriisin dimensiointi voidaan tehdä automaattisestikin. Antamalla käsky

autodim

uudet tulosmatriisit voidaan luoda automaattisesti oikean kokoisiksi. Autodim-komento tarkoittaa tarkasti ottaen sitä, että kaikki uudet symbolitaulun elementit luodaan matriiseiksi (oletusarvo uudelle symbolitaulun elementille on skalaarimuuttuja). Autodim saadaan pois päältä kirjoittamalla sama käsky uudelleen. Autodimin kulloinenkin tila voidaan tarkistaa status-komennolla.

```
Esim.
fread "matri1"           /* luetaan matriisi Mat-istunnon alussa   */
autodim                  /* aktivoidaan automaattinen dimensiointi */
uusimat = rowsum matri1  /* lasketaan matriisien rivisummat        */
autodim                  /* laitetaan autodim pois päältä          */
```

6. TIEDOSTO-OPERAATIOT JA KÄYTTÖJÄRJESTELMÄKOMENNOT

Tiedosto-operaatioilla voidaan tallettaa tiedostoon ja hakea tiedostosta matriiseja. Merkkijonoja ja skalaareita ei sinällään voi tallettaa tiedostoihin, mutta Matissa on mahdollisuus spool-komennon avulla tallettaa kaikki päätteelle tuleva tai näppäimistöltä annettava teksti suoraan tiedostoon. Spool-tiedoston voi suorittaa uudelleen. Tällä tavalla vältytään turhalta kirjoittamiselta.

6.1 MUUTTUJIEN HAKU JA TALLETUS

6.1.1 TEKSTIFORMAATTISET TIEDOSTOT Matriisit voidaan tallettaa tiedostoon komennolla:

fsave <matriisi>

Tällöin matriisi talletetaan ASCII-koodina SCA-formaattiin, tiedoston nimeksi tulee matriisin nimi. Talletuskomennon käytössä tulee olla varovainen, sillä mikäli matriisin nimen mukainen tiedosto on jo olemassa, niin talletuskomento tuhoaa aikaisemman tiedoston ja luo uuden tiedoston tilalle.

SCA-formaatti on omaksuttu Scientific Computing Associationin ekonometrisen ohjelman tekstiaineistoesityksestä. Perusrakenne on sama, mutta havainto- ja aineistojoukot ovat kiinteänimisiä toisin kuin oikean SCA-formaatin yhteydessä.

Formaatti on muotoa:

```
==NAMS
-- Mahdollinen kommenttirivi
OBS COL0 COL1 COL2
    COL3 COL4
END

==DATA
-- Mahdollinen kommenttirivi
Y:1960  1 2 3
        4 5
Y:1961  1 2 3
        4 5
  .   .
  .   .
Y:1989  1 2 3
        4 5
END
```

Muuttujanimet ja aineisto ovat siis erillisinä joukkoinaan. '=='-merkki on joukkotunniste, kiinteät joukkonimet ovat NAMS ja DATA. Nimijoukon on sijaittava ennen aineistojoukkoa. Joukkojen sisäisellä ryhmittelyllä ei ole ahdasta kenttäsidonnaisuutta, vaan havaintonimet ja numeerinen data voidaan kirjoittaa vapaasti joukkojen sisään. Määrien on oltava yhteensopivia, eli DATA-joukon elementtien lukumäärän on oltava jaollinen NAMS-joukon elementtien lukumäärällä. Havaintonimet ovat pakollisia, ja jos niitä ei käytetä, ensimmäinen datasarake ymmärretään nimeksi. Jos käyttää Matin fsave-komentoa, näistä asioista ei tarvitse huolehtia, mutta jos haluaa siirtää aineistoja Mattiin jostakin toisesta ohjelmaympäristöstä, oikeasta formaatista on huolehdittava.

Matriisi luetaan tiedostosta komennolla:

fread <merkkijono>

Tällä käskyllä luetaan lainausmerkkien sisällä annettu tiedosto ja sijoitetaan se samalla nimellä Matin symbolitauluun. Mikäli tiedostonimen mukainen tunnus esiintyy jo symbolitaulussa, lukukäsky keskeytyy virheilmoitukseen.

Tiedostonimen asemesta voi käskyssä käyttää mitä tahansa merkkijonomuuttujaa.

Fsave-komennolla talletettuja muuttujia voidaan käyttää esim. tab- , rep- , table- tai data-ohjelmien yhteydessä.

6.1.2 BINAARIFORMAATTISET TIEDOSTOT Binaaritiedostoissa matriisit talletetaan sisäisten lukuesitysten mukaisesti. Näin saadaan aikaan pakattu talletusmuoto, josta aineiston lukeminen on huomattavasti nopeampaa kuin SCA-formaattisesta tiedostosta. Talletuskomento on muotoa:

bsave <matriisi>

Bsave-komennolla perustetaan matriisinimen mukainen tiedosto, tosin tiedostonimeen liitetään ".b" loppu. Esim. talletettaessa matriisi "matri1" bsave-komennolla, hakemistoon ilmestyy tiedosto "matri1.b".

Bsave-komennolla talletettuja matriiseja voidaan lukea bread-komennolla:

bread <merkkijono>

Bread-komennon yhteydessä tiedostonimeen lisätään automaattisesti '.b'-loppuosa, joten sitä ei saa ilmoittaa lukukäskyn merkkijonossa.

Esimerkki:

```
 bread "binmat1"  /* Luetaan aineiston binmat1.b-nimisestä tiedostosta */
   .    .
   .    .
 bsave binmat1    /* Talletetaan matriisi binmat.b-nimiseen tiedostoon */
```

Myös binaarimuotoisten tiedostojen talletuksessa tulee olla varovainen, sillä samannimiset, jo olemassaolevat tiedostot tuhotaan varoittamatta.

6.2 SPOOL-TIEDOSTOT

Spool-komennolla voidaan tallettaa kaikki komennot ja tulosteet suoraan tiedostoon. Käskyn syntaksi on:

```
spool on      "<tiedostonimi>"   ,
spool on cmd "<tiedostonimi>"    tai
spool on res "<tiedotonimi>"     .
```

Keskimmäinen, cmd-osan (command) sisältävä käsky tarkoittaa, että vain annetut komennot kirjoitetaan tiedostoon ja mahdolliset tulosteet näytetään vain päättellä. Näin cmd-lisäosalla muodostettuja tiedostoja on helppo suorittaa myöhemmin uudestaan esimerkiksi run-komennolla.

Joskus halutaan tulosteita, joissa ei näy suoritettuja käskyjä. Tällöin voidaan spool-tiedosto avata res-optiolla (results).

Spool - tiedostot on suljettava sen jälkeen, kun tiedostoon tulostusta ei enää haluta jatkaa tai tulostustiedostoa halutaan vaihtaa. Sulkemiskomento on:

```
spool off
```

Mikäli Matista poistutaan hallitusti tiedoston loppumerkin (control-D) avulla, kirjoituspuskurit tulostetaan tiedostoon, jonka jälkee se suljetaan. Jos taas ohjelma keskeytetään break-signaalilla, puskureita ei kirjoiteta tiedostoon, joten tulostiedosto jää vaillinaiseksi.

6.3 AJOVIRTATIEDOSTOT JA RUN-KOMENTO

Usein on järkevää kirjoittaa suoritettavat komennot editorilla tiedostoon. Mikäli haluaa "on line" - yhteyden ohjelmaan, kannattanee käyttää spool-komentoa, jotta voi myöhemmin palata kertaamaan suoritettuja komentoja.

Muodostettuja ajovirtatiedostoja voi suorittaa run-komennolla, jonka syntaksi on:

```
run <merkkijono>
```

Kun tiedostossa olevat käskyt on suoritettu, ilmestyy ruudulle lausuma "Ok".

Ajovirtatiedostoja voidaan suorittaa myös siten, että Mattia käynnistet-

täessä komennolla "mat" parametreina annetaan suoritettavien ajovirtatiedostojen nimet. Mikäli ajovirtatiedostot annetaan parametreina ja Mat-istuntoa halutaan jatkaa vielä parametritiedostojen jälkeen, viimeiseksi parametritiedostoksi on määriteltävä "-", joka tarkoittaa standard inputia.

```
Esim: run-komento
      unix-prompt       29:/users/int/et/teksti:
      käynnistä ohjelma 29:/users/int/et/teksti:mat
      aja tiedosto      run "tiedosto"
      jatka Mattia      a = 3 .....

Esim: tiedostot parametreina
      unix-prompt       29:/users/int/et/teksti:
      käynnistä ohjelma 29:/users/int/et/teksti:mat tiedosto -
      jatka Mattia      a = 3 .....
```

Mikäli toinen parametri "-" olisi jätetty pois, Mattia ei enää olisi voitu jatkaa tiedoston "tiedosto" suorittamisen jälkeen.

6.4 **UNIX:-KOMENTO**

Unix: - komennolla voidaan käynnistää Matin sisältä mikä tahansa käyttöjärjestelmäkomento. Tämän komennon avulla on helppo esim. korjata ajovirtatiedostoja tuhoamatta symbolitaulua.

Esim: jos ajovirtatiedostossa on syntaksivirhe, se voidaan korjata käskyllä:

unix: simo <ajovirtatiedosto>

Kun simosta poistutaa virheen korjauksen jälkeen, tilanne palautuu Mattiin täysin siihen tilanteeseen, johon ennen unix: - komentoa jäätiin. Nyt korjattu tiedosto voidaan suorittaa uudelleen:

run <merkkijono>

Unix:-komento on sijoitettu Matin syntaksissa kohtaan, jossa sen käyttö lohkolauseiden yhteydessä (stmt) ei ole mahdollista. Mikäli esimerkiksi luupin sisään haluaan laittaa käyttöjärjestelmäkomennon, on käytettävä komentoa

ux: <merkkijono>.

On huomattava, että ux:-komentoa käytettäessä käyttöjärjestelmäkomento annetaan merkkijonona eli lainausmerkkien sisällä.

7. MUUT KÄSKYT

7.1 PRINT

Print-komennolla voidaan tulostaa merkkijonoja ja skalaarimuuttujia. Print on muotoa:

print <lista> , jossa

<lista> : <lista>
<lauseke>
<lista> , <lauseke>
<lista> , <lauseke>

Esimerkki: tulostetaan n x 1 matriisin matri1 viisi ensimmäistä alkiota, kukin alkio tulostetaan omalle rivilleen taulukkoindeksinsä kanssa

```
for (i=0, i < 5, i++) {
      print i, matri1 [ i,0 ], "\n"
     }
```

Print-lauseessa voi siis olla periaatteessa mielivaltaisen paljon tulostettavia muuttujia.

Matriisi voidaan tulostaa myös vaihtoehtoisella tavalla. Tällöin syntaksi on:

print <matriisi> with <merkkijonovektori> , <lauseke>

Merkkijonovektorissa voi matriisin tulostukseen liittää vapaat muuttujaotsakkeet kullekin sarakkeelle. Riviotsakkeet liitetään tulostuksen ylimmälle riville sellaisenaan kunkin tulostettavan sarakkeen kohdalle. On siis huomattava, että matriisi tulostetaan transpoosina.

Esim: 2x10 matriisi "tulos"

```
dim  tulos [  2, 10  ]
dim  nimet$ [ 10 ] 30
for ( i = 0 ,  i <  10 ,  i ++)  {
  nimet [  i ] =  "Nyt saadaan nimet"
}
tulos =  9999.26
print  tulos  with  nimet$ ,  1
```

```
Tue Sep 15 16:54:25 1987
      Nyt saadaan nimet    9999.3    9999.3
      Nyt saadaan nimet    9999.3    9999.3
      Nyt saadaan nimet    9999.3    9999.3
      Nyt saadaan nimet    9999.3    9999.3
      Nyt saadaan nimet    9999.3    9999.3
      Nyt saadaan nimet    9999.3    9999.3
      Nyt saadaan nimet    9999.3    9999.3
      Nyt saadaan nimet    9999.3    9999.3
      Nyt saadaan nimet    9999.3    9999.3
      Nyt saadaan nimet    9999.3    9999.3
```

Käskyn käyttötapa sevinnee, vaikka nimet$-vektoriin onkin sijoitettu vain vakioinen koenimi. Komento on käyttökelpoinen esimerkiksi silloin, kun halutaan tulostaa pitkiä raportointikelpoisia muuttujanimiä. Yleensä raportointi kannattaa tehdä rep-ohjelmalla, mutta jotkut pienimuotoiset tulostukset voidaan toteuttaa tällä erityiskomennolla.

7.2 **INPUT JA READ**

Input-lauseella voidaan lukea aineistoja muuttujiin samoin kuin read-lauseellakin. Erona näissä kahdessa lauseessa on se, että input lukee pyydettyjä arvoja aina standard inputista eli näppäimistöltä, kun taas read lukee muuttujiin arvoja käsiteltävänä olevasta tiedostosta. Mikäli käsiteltävänä tiedostona on standard input, näillä kahdella lauseella ei ole mitään eroa. Jos taas käsiteltävän tiedostona on ajovirtatiedosto, sinne sijoittettua dataa voidaan lukea tiedostosta read-lauseella. Lread on muuten vastaava kuin read, mutta sillä luetaan merkkijonona kokonainen rivi eikä vain yhtä merkkijonoa. Tämä mahdollistaa monisanaisten virkkeiden tai otsakkeiden lukemisen tiedostosta.

Syntaksit ovat:

```
input <lista>
read  <lista>
lread <lista>

<lista>   :    <lista>
               merkkijonomuuttuja
               skalaarimuuttuja
               <lista> , skalaarimuuttuja
               <lista> , merkkijonomuuttuja
```

Esim: määritellään aliohjelma, joka lukee näppäimistöltä
matriisiin matri1 lukuarvot

```
dim matri1 [ 10, 1 ]
proc readto() {

    for (i=0, i < 10, i++) {
      print "Anna luku ",i," : "
      input matri1 [ i, 0 ]
   }
 }
```

Esimerkki: edellisen tyyppinen aliohjelma, joka lukee datan käsiteltävänä olevasta tiedostosta. Jos tämän aliohjelman kutsu sijaitsee tiedostossa ajo ja kutsu suoritetaan komennolla run "ajo", matriisin matri1 arvot luetaan tiedostosta ajo. Mikäli edellisen aliohjelman readto kutsu olisi tiedostossa ajo, matriisin matri1 arvot luettaisiin joka tapauksessa näppäimistöltä.

```
proc readto2() {
   print "Luen havaintosi\n"
   for (i=0, i < 10, i++) {
      read matri1[i,0]
    }
 }
```

7.3 SYMBOLS

Symbols-komennolla voidaan tulostaa käyttäjän määrittelemät symbolitaulun elementit. Syntaksi on:

symbols

Ne symbolitaulun elementit, joiden edessä ei listauksessa ole '>'-merkkiä, ovat hashing-funktion aikaansaamia yhteentörmäyksiä, jotka linkitetään symbolitaulun samaan kohtaan peräkkäin. Tämä hidastaa symbolien hakua, jos yhteentörmäyksien määrä on suuri. Käytännön kokemukset ovat kuitenkin osoittaneet, että yhteentörmäyksiä ei esiinny liikaa.

7.4 STATUS

Status-komennolla voidaan tulostaa eräitä matin voimassa olevia määrittelyjä, esim. piirtojänteen pituus, vallitseva piirtotulostuslaite (Canon tai pääte) ja autodimin tila (päällä tai pois). Ykköset tarkoittavat, että kyseessä oleva ilmiö tai laite on aktivoitu. Nollat tarkoittavat päinvastaista.

7.5 ALKUASETUKSET OHJELMAA KÄYNNISTETTÄESSÄ

Mattia käynnistettäessä lähetetään päätteelle clear-sekvenssi, joka puhdistaa näyttöpuskurin ja siirtää kursorin vasempaan ylälaitaan. Joissakin tilanteissa tämä piirre saattaa olla kiusallinen. Esimerkiksi jos näytöllä on tulosteita, jotka halutaan mat-istunnon aikana pitää näkyvillä, päätepuskurin oletusarvoinen tyhjennys on kytkettävä pois päältä. Mat käy aina ohjelmaa käynnistettäessä tutkimassa, löytyykö prosessin omistajan (käyttäjän) kotihakemistosta ".matrc"-nimistä tiedostoa. Mikäli tiedostoa ei löydy, mat käynnisetetään oletusarvoilla. Mikäli tiedosto on olemassa, siihen kirjoitetut käynnistysasetukset luetaan ja aktivoidaan. Mahdolliset asetukset ovat:

Käskysana	Arvo	Oletusarvo	Selitys
flush	0/1	1	Näyttö tyhjennetään ohjelman alussa,
poet	0/1	0	Satunnainen mietelause tulostetaan näyttöön

7.6 TULOSTUSTARKKUUDEN SÄÄTÄMINEN

Jos käyttäjä haluaa tulosteeensa oletusesityksestä poikkeavalla tarkkuudella, on käytettävä precision-lausetta. Syntaksi on:

precision <lauseke> , <lauseke>
  tai
string precision <lauseke>.

Precision-tarkkuuskomennolla säädetään numeerisen aineiston tulostustarkkuutta: ensimmäisellä parametrilla ilmoitetaan tulostuskentän pituus ja toisella tulostettavien desimaalien lukumäärä. Parametrien arvoilla -1,-1 oletusarvot saadaan takaisin.

Merkkijonojen tulostuskentän pituutta säädetään string precision-lauseella. Oletusarvo saadaan takaisin parametrin arvolla 0.

8. REGRESSIOANALYYSI

Regressioanalyysirutiini on erillinen ohjelmansa, jota Mat kutsuu pyydettäessä. Regressioanalyysi suoritetaan pivotoimalla alimatriiseittain.

Aineisto poimitaan jo Mattiin luetuista matriiseista sarakkeittain. Viittaukset matriisin sarakkeisiin eli muuttujiin ovat muotoa:

< matriisi >.< sarakeotsake >.

Tämän tyyppistä ilmaisua merkitään <savi>:lla, jossa <savi> tarkoittaa sarakeviittausta.

Itse regressiokutsu on muotoa:

```
reg dep  <savi> ,                    @
    ind  <savi> ... <savi> ,         @
    span <lauseke> to <lauseke> ,    @
    rem  <merkkijono>
```

Dep tarkoittaa selitettävää, ind selittäviä muuttujia ja span mukaanotettavien havaintojen jännettä. Rem-lauseeseen voi liittää mallin nimen tai muun kommentin. Rem-lauseella on oma erityismerkityksensä, mikäli saman Mat-istunnon aikana estimoidaan useita malleja, joissa on samoja selittäviä muuttujia. Tällöin ei ole mahdollista pelkästään selittäjän nimen perusteella päätellä sitä, mihin malliin _B1-vektorissa palautettava regressiokerroin kuuluu. Mikäli rem-lauseella välitetään regressio-ohjelmaan kokonaisluku (tosin merkkijonona, koska rem-lause on syntaksin mukaisesti merkkijonoparametrinen), eri malleihin kuuluvat selittäjät voidaan erotella toisistaan tämän kokonaisluvun avulla. Mikäli rem-lauseessa annetaan kokonaisluku, se välittyy regressiolohkosta takaisin _B0-vektorissa. Tällaisen järjestelyn ansiosta on mahdollista käyttää estimoituja kertoimia suoraan Matin yhteydessä.

Vaikka dokumenttitekstissä regressiolause on jaettu useammalle riville, lause voidaan kirjoittaa yhdelle riville. Mikäli lause halutaan esim. selkeyssyistä kirjoittaa useammalle riville, niin tämäkin on mahdollista, jos käytetään matin yleistä komennonjatkomerkkiä '@' .

Ohjelma tallettaa automaattisesti residuaalit vektoriin def_res ja sovitteet vektoriin def_pred (default residuals ja default predictions). Sovitteiden ja jäännösten kohdentumista voidaan muuttaa ennen reg-komentoa annettavalla res- tai pred-komennoilla. Käskyt ovat muotoa:

```
res  <savi>
pred <savi> ,
```

jossa <savi> tarkoittaa epäsuoraa sarakeviittausta.

Esimerkki: tiedostoon on talletettu matriisi rtest, jolle suoritetaan regressioanalyysi. Tulosteet ovat seuraavat:

```
bread "rtest"
rtest : 29 rows, 9 cols

name print rtest

rtest
Row names:
```

```
              00              01              02              03
              04              05              06              07
              08              09             010             011
             012             013             014             015
             016             017             018             019
             020             021             022             023
             024             025             026             027
             028
Col names:
            CONS             AB1I            CB1I             TR1
             IA1             POI1             LV1          ROCFBT1
             DR1

 reg  dep  rtest.AB1I,                          @
      ind  rtest.CONS rtest.CB1I rtest.TR1, @
      span 0 to 28,                           @
      rem "malli1"

        Variable          Mean           SST
      rtest-CONS             1             0
      rtest-CB1I    0.03424138     0.5360973
       rtest-TR1     0.9328276     7.785734

 OLS estimation

dependent variable: rtest.AB1I

Independent var    Est coeff.   Standard err   t-stat
      rtest.CONS  0.3634185     0.04443477      8.178697
      rtest.CB1I -0.5173157      0.1550151     -3.337195
      rtest.TR1  -0.02618819     0.04067674    -0.6438124

Number of observations:       29
R-squared                0.299892
Standard error           0.111259
F-statistics             0.127968
```

9. VIRHEILMOITUKSET

Matissa on useita virheilmoituksia, jotka on listattu selityksineen alla.

Virheilmoitus	Selitys
syntax error	Käyttäjän antama lause

	ei vastaa kielioppia
fatal error, approaching CRASH	Muistialueylivuoto, joka on seurausta ohjelman tekijän huolimattomuudesta
floating point exception	Ns. kontrolloimaton rekisteriylivuoto, esimerkiksi nollalla jakaminen, jonka mahdollisuus on jäänyt ohjelman tekijältä huomaamatta
out of range	Taulukkorajat on ylitetty
number of columns don't match	Kahden matriisin sarakkeiden lukumäärät poikkeavat toisistaan
dimension error in cov	Kovarianssimatriisin dimensio on laiton
dimension error in corr	Korrelaatiomatriisin dimensio on laiton
kill matrix before cov	Tulosmatriisi on epäsopiva kovarianssilaskennassa. Tulosmatriisi on poistettava kill-komennolla ennen cov-funktion suoritusta.
kill matrix before dev	Sama kuin edellinen, mutta koskee dev-funktiota
kill matrix before corr	Sama kuin edellä, mutta koskee corr-funktiota
only for square matrix	Toiminto on tarkoitettu ainoastaan neliömatriisille
pivot element is zero	Pivotoitavan matriisin lävistäjäelementti on nolla
stack too deep	Pino on täynnä, lauseke tai suoritettava lohko on liian iso Matille
stack underflow	Pinon alivuoto, ohjelmatekijän tekemä virhe, ilmoitus on harvinainen
out of memory	Ohjelma ei ole saanut systeemiltä lupaa käyttäjän toivomiin tilavarauksiin
call nested too deeply	Liian monta sisäkkäistä funktiokutsu, funktiopinon ylivuoto
(proc) returns no value	Käyttäjän määrittelemä funktio ei palauta arvoa
(proc) returns value	Käyttäjän proseduuri palauttaa arvon
not enough arguments	Käyttäjän määrittelemässä funktiossa tai proseduurissa tarvitaan enemmän argumentteja kuin mitä funktiokutsussa

	on annettu
division by zero	Nollalla jako, mikä aikaansaisi rekisteriylivuodon
end of file encountered	Syötetiedosto on loppunut kesken lohkolauseen
non-number read into	Ei-numeerista alkiota on yritetty lukea numeeriseen muuttujaan
program too big	Kone on täynnä
size error in matrix	Matriisi on epäsopiva kutsuttuun matriisilauseeseen
matrix error in inverse	Matriisi on väärän kokoinen kääntämiseen
matrix is singular	Matriisi on singulaarinen
not a square matrix	Matriisi ei ole nelömatriisi, vaikka sen pitäisi olla
illegal filename	Käyttäjän antamaa tiedostoa ei pystytä avaamaan
only one spool file	Käyttäjä on yrittänyt avata kaksi spool-tiedostoa yhtä aikaa
illegal name	Laiton tiedostonimi
scanning the universe, in vain	Käyttäjän antamaa symbolia ei löydy
not possible yet, use indirect..	Toiminto on mahdollista ainoastaan epäsuorilla sarakeviittauksilla
Nobody home in name area	Matriisin nimialuetta ei jostakin syystä ole allokoitu
Can't open	Tiedostoa ei voida avata
data file not found	Tiedostoa ei löydy
can't find matrix	Matriisia ei löydy symbolitaulusta
symbol exists in symbol table	Luotava symboli on jo olemassa
forking error in draw	Graph-ohjelmaa ei pystytä kunniallisesti hajauttamaan omaksi prosessikseen
No stetson solution	Regressio-ohjelma ei saa kertoimia palautettua emo-ohjelmaan
Target matrix has to be a new...	Lauseen tulosmatriisia ei saa olla olemassa ennen toiminnon suoritusta
Column structures don't match	Sarakkeet eivät ole yhteensopivia
parameter not found	Emo-ohjelma on epäonnistunut parametrien palauttamisessa lapsiprosessilta
unable to read	Mat ei löydä etsittävää symbolia
undefined matrix (use dim...)	Matriisia ei ole julistettu
strange parameter ref. %...	Laiton parametri käyttäjäfunktiossa
name too long	Symbolin nimi on liian pitkä
can't create indirect assignment	Mat ei pysty luomaan epäsuoraa

	sarakeviittausta, luultavasti sen takia, koska siihen tarvittavaa matriisia ei ole olemassa
unknown sys. var	Käyttäjä kutsunut systeemimuuttujaa, jota ei ole olemassa
unknown keyword	Käytetty käskysanaa, jota ei ole olemassa
missing quote	Merkkijonon lopusta puuttuu lainausmerkki
string too long	Merkkijono on liian pitkä
end of file encountered, missin...	Kommentti jatkuu tiedosto loppuun ilman, että sitä lopetettaisiin
used outside definition	Return-käskysanaa käytetty funktion ulkopuolella

Edelliset olivat virheitä, jotka aikaansaavat käsittelyssä oleva tiedoston loppuosan ohittamisen ja siirtymisen seuraavaan suorituslistassa olevaan tiedostoon. Seuraavat ilmoitukset ovat varoituksia, joiden edessä lukee aina warning. Ne eivät keskeytä toimintoja, vaan virhelause suoritetaan niin pitkälle kuin mahdollista ja suoritusta jatketaan seuraavsta lauseesta. Esimerkiksi jos matriisisijoituksessa yritetään matriisia sijoittaa erikokoiseen matriisiin, erisuuruudesta annetaan varoitus, mutta vasemmasta yläkulmasta sijoitetaan niin paljon kuin pienempi matriisi antaa myöten.

unequal sizes	Matriisilauseen matriisit ovat erikokoisia
division by zero	Nollalla jako, ja tähän kohtaan sijoitetaan todennäköisesti puuttuva havainto (1e+037).
best way to galaxy	Käyttäjä voi mennä itseensä
assignment to first row	Lauseen tulos on nx1 rivivektori, joka sijoitetaan tulosmatriisin ensimmäiseksi riviksi
assignment to first column	Sama kuin edellä, mutta kyseessä on sarakevektori
first row processed in	Ainoastaan ensimmäinen rivi käsitellään
unexpected end of file	Odottamaton tiedoston loppu, suoritus kuitenkin jatkuu
domain error in log	Logaritmifunktioon on tuputettu nollaa tai negatiivista lukua, mahdollinen esimerkiksi logdif-funktiossa

10. KUVIOIDEN PIIRTÄMINEN

Matista voidaan suoraan käynnistää graph-ohjelma. Tällöin Mat luo väliaikaisen tiedoston, jossa välitetään graphille kuvion tarvitsema aineisto. Väliaikainen tiedosto tuhotaan automaattisesti piirtämisen jälkeen. Piirtäminen käynnistetään draw-komennolla, ja draw-käskysanan perään voidaan liittää epäsuorista sarakeviittauksista koostuva lista piirrettävistä sarjoista. Matin graph-liitäntä on toistaiseksi suunniteltu ainoastaan aikasarjojen piirtämistä varten, eikä esim. kahden muuttujan välistä hajontakuviota voida vielä tehdä. Mikäli haluaa piirtää tämän tyyppisiä kuvioita, kätevin tapa on tallettaa piirrettävät muuttujat fsave-käskyllä tiedostoon ja käyttää graphia suoraan talletetussa tiedostossa olevalla aineistolla. Matin binaaritiedostot ovat yhteensopivia graphin, tablen (ja Mimer-tietokannan tulostustiedostojen) kanssa.

Mikäli halutaan piirtää vain osa havainnoista, jännettä voidaan supistaa span-komennolla. Määritelty aikaväli on erikseen aktivoitava span on -komennolla. Vastaavan deaktivoinnin saa aikaan span off -komennolla.

```
span 0 to 15              /* 16 ensimmäistä havaintoa piirtojänteeksi */
span on                   /* aktivoidaan jännemäärittely              */
draw koe.ensi koe.toinen /* piirretään kaksi sarjaa samaan kuvaan    */
```

Mikäli kuva halutaan Canonin laserkirjoittimelle, paperitulostusmuoto on aktivoitava käskyllä

```
canon on.
```

Tämän komennon jälkeen kaikki draw-komennot kohdistuvat Canon-kirjoittimelle. Kuvat saadaan kohdistumaan jälleen päätteelle, mikäli käytetään komentoa

```
canon off.
```

Jos haluaa piirtää viivan (line) asemesta pylväitä (bar), voidaan piirtomuotoa muuttaa komennolla

```
type bar
```

ennen draw-komentoa. Pylväät saa takaisin viivaksi käskyllä

```
type line.
```

Mikäli piirrettäviä havaintoja on yli 255 kappaletta, on aktivoitava laaja graph komennolla

```
Graph.
```

Tämän jälkeen Mat kutsuu graph-versiota, jossa laajemmat tilanvaraukset

mahdollistavat suurien aineistojen piirtämisen. Laajaa graphia ei kannata käyttää turhaan, sillä se rasittaa konetta huomattavasti enemmän kuin suppea graph. Deaktivointi suoritetaan kirjoittamalla Graph-komento uudelleen.

11. OHJELMAESIMERKKEJÄ

Tässä luvussa esitellään pääpiirteittäin muutamia ohjelmasovelluksia, joita Matilla on toteutettu. Ensimmäinen esimerkki on lyhyt kuvaus tulkin käytöstä ja matriisioperaatioiden avulla tehdystä regressioanalyysista. Aineisto on peräisin Helsingin Yliopiston tilastotieteen laitoksen Data-analyysikurssin harjoitustehtävästä. Toisena esimerkkinä on panos-tuotos -malli, jonka Mat-sovitus on Eija Kaupin tekemä. Matilla on myös rakennettu mm. Markku Rahialan kulutusmalli, jonka Mat-toteutus on niin ikään Eija Kaupin käsialaa.

11.1 REGRESSIOANALYYSI MATRIISIOPERAATIOILLA

```
/* Tiedosto /users/int/et/jakki/mac7b/demo3, ET 25.8.88 */

spool on "tulos1" /* Avataan tulosvirtatiedosto */
fread "dmda1"     /* Luetaan havaintomatriisi */
name print dmda1  /*  Tulostetaan rivi- ja sarakeotsakkeet ja todetaan,
                      että vakiota ei matriisissa ole  */
dim dmda2[ 10,6 ] /*  Julistetaan matriisi, jossa on yksi sarake
                      enemmän kuin dmda1-matriisissa. Vakio voidaan sijoittaa
                      tähän matriisiin  */
dmda2 = dmda1     /*  Sijoitetaan matriisi toiseen. Vaikka ne ovat eri
                      kokoisia, sijoitusta suoritetaan niin paljon kuin
                      mahdollista. Epäsopivuudesta annetaan varoitus.  */
dmda2 5 col name = "vakio" /*  Nimetään viimeinen sarake vakioksi  */
dmda2.vakio = 1   /*  vakion arvo on 1  */
change dmda2 0 col , dmda2 5 col  /*  Vaihdetaan vakio matriisin
                                      ensimmäiseksi sarakkeeksi  */
change dmda2 4 col, dmda2 5 col   /*  Vaihdetaan selitettävä muuttuja
                                      matriisin viimeiseksi
                                      sarakkeeksi  */

/*  Vakiolla täydennetty havaintomatriisi on nyt valmiina  */

autodim           /*  Kytketään selaaja tilaan, jossa se palauttaa
                      uuden symbolin tyyppinä matriisin.
                      Oletusarvo on skalaari.  */

x    =  dmda2     /*  Käytetään hieman siistimpää
```

```
                    havaintomatriisimerkintää  */
trx   =  trn( x ) /* Havaintomatriisin transpoosi */
xx    =  trx * x  /* Laajennettu momenttimatriisi */

/*  Pivotoidaan laajennettu momenttimatriisi eri lävistäjäelementtien
    suhteen  */

px1 = pivot  xx, 0
px2 = pivot px1, 1
px3 = pivot px2, 2
px4 = pivot px3, 3
px5 = pivot px4, 4

print px5  /*  Tulostetaan kaikkien lävistäjäelementtien suhteen
               pivotoitu momenttimatriisi. Regression tulokset
               on luettavissa matriisista  */

cormat = corr( dmda1 ) /*  Lasketaan vielä muuttujien välisiä
                           korrelaatioita                    */
print cormat           /* Tulostetaan korrelaatiomatriisi      */
symbols                /*  Katsotaan, mitä elementtejä
                           symbolitauluun on ilmestynyt      */
spool off              /* Suljetaan spool-tiedosto           */
```

Edelliset käskyt voidaan siis suorittaa joko suoraan antamalla tulkille standard inputiin komennot, toinen vaihtoehto on kirjoittaa ne tiedostoon ja suorittaa ne esimerkiksi run-komennolla. Alussa aktivoitu spool-komento saa aikaan sen, että kaikki tulokset näkyvät paitsi näytössä myös spool-tiedostossa. Jos spool käskynä olisi ollut "spool on cmd", ainoastaan komennot olisivat tulostuneet tiedostoon. Tämä vaihtoehto on käyttökelpoinen silloin, kun halutaan kokeilla tulkissa erilaisia komentoja ja silti tallettaa ne myöhempää käyttöä tai uudelleensuoritusta varten. Tiedosto tulos1 on listattu alla. Kuten tuloksista voidaa huomata, kommenttien tiedostotulostuksessa on pieni virhe, merkistä '/*' tulee '/*/'. Virhe ei juuri haittaa ohjelman käyttöä.

```
 fread  "dmda1"   /*/ Luetaan havaintomatriisi */
dmda1 : 10 rows, 5 cols
 name  print  dmda1 /*/ Tulostetaan rivi- ja sarakeotsakkeet ja todetaan,
                        että vakiota ei matriisissa ole  */

dmda1
Row names:
              1              2              3              4
              5              6              7              8
              9             10
Col names:
             X1             X2             X3             X4
              Y
```

```
 dim  dmda2 [  10 , 6  ] /*/  Julistetaan matriisi, jossa on yksi sarake
                      enemmän kuin dmda1-matriisissa. Vakio voidaan sijoittaa
                      tähän matriisiin  */
 dmda2 =  dmda1 /*/  Sijoitetaan matriisi toiseen. Vaikka ne ovat eri
                      kokoisia, sijoitusta suoritetaan niin paljon kuin
                      mahdollista. Epäsopivuudesta annetaan varoitus.  */
mat: warning: unequal sizes
 dmda2  5   col  name = "vakio" /*/  Nimetään viimeinen
                                     sarake vakioksi  */
 dmda2.vakio  =  1   /*/  vakion arvo on 1  */
 change  dmda2  0   col ,  dmda2  5  col /* / Vaihdetaan vakio matriisin
                                     ensimmäiseksi sarakkeeksi  */
 change  dmda2  4   col ,  dmda2  5  col /*/  Vaihdetaan selitettävä muuttuja
                                     matriisin viimeiseksi
                                     sarakkeeksi  */

/*/  Vakiolla täydennetty havaintomatriisi on nyt valmiina  */

 autodim /*/  Kytketään selaaja tilaan, jossa se palauttaa
                  uuden symbolin tyyppinä matriisin.
                  Oletusarvo on skalaari.  */

 x =   dmda2 /*/  Käytetään hieman siistimpää
                   havaintomatriisimerkintää  */
 trx =   trn ( x )  /*/ Havaintomatriisin transpoosi */
 xx =   trx * x /*/ Laajennettu momenttimatriisi */

/*/  Pivotoidaan laajennettu momenttimatriisi eri lävistäjäelementtien
   suhteen  */

 px1 =  pivot  xx ,  0
 px2 =  pivot  px1 ,  1
 px3 =  pivot  px2 ,  2
 px4 =  pivot  px3 ,  3
 px5 =  pivot  px4 ,  4

 print  px5 /*/  Tulostetaan kaikkien lävistäjäelementtien suhteen
              pivotoitu momenttimatriisi. Regression tulokset
              on luettavissa matriisista  */
```

```
px5
              vakio          X2          X3          X4          X1
 vakio       670.429    -7.3668    -9.34221    -8.38825    -6.43683
    X2       -7.3668   0.0851544    0.098206   0.0879968   0.0704299
    X3      -9.34221    0.098206    0.150125    0.124994   0.0867951
    X4      -8.38825   0.0879968    0.124994     0.12691   0.0773344
    X1      -6.43683   0.0704299   0.0867951   0.0773344   0.0629721
     Y       197.349    -2.65684    -2.32704    -1.99964     -2.8035

                   Y
 vakio       -197.349
    X2        2.65684
    X3        2.32704
    X4        1.99964
    X1         2.8035
     Y        26.7347


 cormat =  corr ( dmda1 ) /*/  Lasketaan vielä muuttujien välisiä
                        korrelaatioita  */
 print  cormat /*/ Tulostetaan korrelaatiomatriisi */

cormat
                X1          X2          X3          X4           Y
    X1           1    -0.81952   -0.274862   -0.110993    0.721942
    X2    -0.81952           1   -0.0236747   -0.0714001   -0.309034
    X3   -0.274862   -0.0236747           1    -0.58633    -0.29853
    X4   -0.110993   -0.0714001    -0.58633           1     -0.1959
     Y    0.721942    -0.309034    -0.29853     -0.1959           1


 symbols /*/  Katsotaan, mitä elementtejä
                    symbolitauluun on ilmestynyt  */
>cormat_dev_mean  : matrix of  1,5
>px1  : matrix of  6,6
>px2  : matrix of  6,6
>px3  : matrix of  6,6
>px4  : matrix of  6,6
>px5  : matrix of  6,6
>cormat_mean  : matrix of  1,5
>£CMD$  :  (0/256)
>cormat  : matrix of  5,5
>_B0  : matrix of  1,256
>_B1  : matrix of  1,256
>_B2  : matrix of  1,256
>x  : matrix of  10,6
>dmda1  : matrix of  10,5
>dmda2  : matrix of  10,6
>xx  : matrix of  6,6
>cormat_dev  : matrix of  1,5
```

```
 £EDIT$  :  (0/2025)
>trx   : matrix of  6,10
 spool  off /*/ Suljetaan spool-tiedosto */
```

Symbolitaululistauksesta huomataan, että se sisältää muitakin kuin käyttäjän julistamia elementtejä. Osa vieraista elementeistä syntyy suorituksen aikana, ja osa sijoitetaan tauluun jo ohjelmaa käynnistettäessä. Symbols-komento ei näytä kaikkia symbolitaulun elementtejä. Esimerkiksi funktioita ei näytetä, koska niiden suuri lukumäärä häiritsisi symbolitaululistauksen lukemista. Ne symbolit, joiden edessä ei ole '>'-merkkiä, ovat hashing-funktion aikaansaamia yhteentörmäyksiä. Niitä on suhteellisen vähän, joten Matin hashing-funktion voidaan todeta selviytyvän tehtävästään kiitettävästi. Korrelaatiomatriisia laskettaessa allokoidaan tauluun automaattisesti '_dev'-loppuinen keskihajontamatriisi, jota tarvitaan korrelaatiota laskettaessa. Hajontaa laskettaessa taas tarvitaan keskiarvoja, jotka sijoitetaan '_mean'-loppuiseen matriisiin. Näin ollen symbolitauluun ilmestyy edellisen esimerkin aikana matriisi 'cormat_dev_mean'. Taulussa on myös matriisi 'cormat_mean', joka allokoidaan korrelaatiomatriisin tarpeisiin Keskiarvomatriisi lasketaan tässä tapauksessa kahteen kertaan, joten eri funktioiden välisessä kommunikaatiossa on siis rationalisoimisen varaa. Tässä versiossa järkeistämistä ei vielä ole ajateltu loppuun asti. Eräs vaihtoehto olisi käyttää väliaikaisia tilavarauksia, mutta koska unix-prosessin puitteissa tilavarauksen purkaminen ei ole täydellistä, on päädytty käytetyn tilan ilmoittamiseen. Alaviivalla alkavat elementit ovat ns. systeemimuuttujia. Esimerkiksi matriisiin _B1 sijoitetaan varsinaisen regressiokutsun palauttamat kertoimet. Risuaitamerkillä alkavat symbolit ovat vakioita. Kaikkia vakioita ei listauksessa näytetä.

11.2 **PANOS-TUOTOS -MALLI**

Panos-tuotos -mallin Mat-koodi on tässä kappaleessa käyty pääpiirteittäin läpi. Koodia on osin supistettu, koska sen täydellinen listaus on niin pitkä, että se vaatisi oman dokumenttinsa. Tässä yhteydessä tarkoituksena on antaa kuva siitä, minkälaisia sovelluksia Matilla voidaan tehdä. Panos-tuotos -mallin ideana on se, että jos tiedetään kysyntä sektoreittain, hyödykeryhmittäin tai pääkysyntäkomponenteittain, voidaan kiinteiden muunnosmatriisien avulla approksimoida kysynnän aikaansaamaa tuotantoa. Kysynnän avulla siis tarkastellaan tarjonnan muodostumista.

```
/* Aluksi luetaan tiedosto, jossa annetaan ennustevuodet */

fread "enn_vuosi"
enn = enn_vuosi.venn [ 0 ]
vlkm = enn_vuosi.lkm [ 0 ]
  .     .
  .     .
```

```
/*  Tämän jälkeen luetaan varsinainen aineisto, joka
on jaoteltu osin historiallisista ja osin mukavuussyistä
useisiin eri tiedostoihin.  */

fread "kuja"            /* jaettava, yksityinen kulutus */
fread "kumu"            /* muunnosmatriisi, yksityinen kulutus */
  .      .
  .      .
fread "inmu"            /* muunnosmatriisi, investoinnit */
fread "eskolta"         /* koko talouden kulutukset */
  .      .
  .      .

/*  Laskennassa tarvitaan useita eri matriiseja, jotka
dimensioidaan käsiteltävien vuosien lukumäärän mukaan  */

dim kutu [ rr, rp ], jutu [ rr, rp ]
  .      .
  .      .
dim kujat[11,rp], jujat[4,rp]

/*  Annetaan välitulostus käyttäjän rauhoittamiseksi  */

print "\n\n  Kysynnän ennustetietojen käsittely ja "
print "\n tuotantomatriisin muodostaminen ... "

/*  Ajetaan ohjelmat-alihakemistossa oleva Mat-syötettä sisältävä
tiedosto mu_tasot. Tässä tiedostossa määritellään myöhemmin kutsuttavia
käyttäjäfunktiota  */

run "ohjelmat/mu_tasot"

pros_to_taso()                          /* enn. pros. tasoiksi */
ap1 = rowsum kuja                       /* yksityinen kulut. yht. */
ap3.COL0 = inja.MTRAK + inja.MVRAK      /* investoinnit, lukuunottamatta */
 .      .
 .      .
ap6 = ap6 - ap7                     /* tavaroiden vienti */
tasaa()                             /* kokonaisenn. eron jako */
uud_pros()                          /* uudet ennusteprosentit */
muunnokset()                        /* "kysyntä tuotannoksi"  */
 .      .
 .      .
print "\n  Tilastovirheen käsittely ...  "
run "ohjelmat/til_virhe"
tuotanto_virhe()                        /* tilastovirheen osuus tuot. */
regr()                                  /* regr. anal tv:n ennustamiseen */
print "\n Tuloksen kokoaminen ... "
run "ohjelmat/kokoa_tulos"
tulos_matr()                            /* tulosmatriisin kokoaminen */
tilastovirhe()                          /* tv:n kokoaminen tulostukseen */
```

```
  .     .
  .     .
bsave kutu                          /* yksit. kulut. aiheuttama tuotanto */
```

Matin käyttäjä voi siis ohjelmoida tulkkia vastaavasti kuin varsinaisilla ohjelmointikielillä. Mat on hyvin mukautuvainen: toisaalta sitä voi käyttää helposti perusoperaatioihin, mutta toisaalta sen käyttöä voi laajentaa hyvinkin pitkälle. Jos funktiot ja proseduurit kokoaa omiksi tiedostoikseen, päätason Mat-koodista tulee suhteellisen selkeää. Tiedosto ohjelmat/mu_tasot, joka sisältää joitakin aliohjelmia, näyttää seuraavalta:

```
proc pros_to_taso() {
   alku = enn - 1980
   loppu = alku + vlkm - 1
   for ( i = alku, i <= loppu, i++ ) {
       kuja.EJT[i] = kuja.EJT[i-1] * (putfppe.EJT[i] / 100 + 1 )
       kuja.VAJA[i] = kuja.VAJA[i-1] * (putfppe.VAJA[i] / 100 + 1 )
           .           .
           .           .
       kuja.ASUM[i] = kuja.ASUM[i-1] * (putfppe.ASUM[i] / 100 + 1 )
       putfp.QKTKTP[i] = putfp.QKTKTP[i-1] * ( putfppe.KTKTP[i]/100+1 )
       vija.UKMS[i] = 0 - putfp.QUKMS[i]
    }
}

proc tasaa() {
    ero.COLO =  eskolta.YKU - ap1.COLO
           .           .
           .           .
    alku = enn - 1980
    for ( i = alku, i < alku + vlkm, i++ ) {
        /* yksityissektori */
        for ( j = 0, j < 8, j++ ) kuja[i,j] = kuja[i,j] + @
            ero.COLO[i] * kuja[i-1,j] / eskolta.YKU[i-1]
        putfp.QKTKTP[i] = putfp.QKTKTP[i] + ero.COLO[i] * @
                          putfp.QKTKTP[i-1] / eskolta.YKU[i-1]
        putfp.Q2KUNN[i] = putfp.Q2KUNN[i] + ero.COL1[i] * @
                          putfp.Q2KUNN[i-1] / eskolta.JKU[i-1]
           .           .
           .           .
        /* vienti, palvelut */
        for ( j = 10, j < 12, j++ ) vija[i,j] = vija[i,j] + ero.COL6[i] * @
            vija[i-1,j] / ( eskolta.VKUp[i-1] - vija[i-1,12] )
        putfp.TAVV[i] = 0
        for ( k = 0, k < 10, k++ ) putfp.TAVV[i] = putfp.TAVV[i] + vija[i,k]
        putfp.PALVV[i] = vija.KULJV[i] + vija.MUUPV[i] + vija.UKMS[i]
        putfp.VIENTI[i] = putfp.TAVV[i] + putfp.PALVV[i]
    }
```

Matilla saa siis aikaan paitsi lyhyitä matriisilauseita myös pitkiä risuaitaviidakoita, jotka useissa sovelluksissa ovat välttämättömiä.

12. OHJELMAN TOIMINTAPERIAATTEISTA

12.1 OHJELMOINTITYÖKALUISTA

Matissa ohjelmarakentamisen perustyökaluina ovat olleet Yacc-metakääntäjä (yet another compiler compiler) ja C-ohjelmointikieli. Yacc-ohjelmointiapuvälineen kehitti vuonna 1972 Steve Johnson, joka ajan muotivirtauksia kommentoiden nimesi metakääntäjänsä Yacc:iksi (suomeksi: ja taas yksi metakääntäjä). Vaikka Yacc ei alun perin olekaan public domain-ohjelma, siitä on ilmestynyt kokemusten mukaan täysin yhteen sopiva pd-tuote. Näin ollen Yacc-ohjelmointiympäristö on saatavissa myös esimerkiksi mikrotietokoneympäristöön. Yacc:n käyttökelpoisuuden ydin on siinä, että sen avulla on helppo määritellä selkeitä kielioppeja. Myös kieliopin muutokset ja lisäykset sujuvat vaivatta, joten Yacc:lla tehtyjen kielioppien muuttamiskynnys on matalampi kuin puhtailla C-kielisillä kieliopeilla.

Yacc-työskentelyn perusideana on se, että ohjelmoija määrittelee tarvitsemansa kieliopin Yacc:n syntaksin mukaisesti. Tämä kielioppimäärittely käännetään C-lähdekoodiksi, joka voidaan kääntää ja linkata muiden C-lohkojen kanssa toimivaksi ohjelmaksi.

Ohjelmointikielen lähdekoodin peruselementtejä ovat selaaja ja jäsentäjä. Selaaja etsii syötteestä (tulkattavasta Mat-käyttäjän koodista) kielioppiin kuuluvia symboleja, jotka se välittää eteenpäin. Syötteestä poimitut symbolit annetaan jäsentäjälle, joka kokoaa useasta eri symbolista kokonaisia lauseita ja suorittaa ne. Mikäli syötteestä poimitulle lauseelle ei löydy vastinetta ohjelman kieliopista, kysymyksessä on syntaktinen virhe eli "syntax error".

Yacc on äärellinen tilapinoautomaatti, joka jäsentää syötettä LALR-periaatteen mukaisesti (look ahead left right parser). Tämä tarkoittaa sitä, että syötteessä ei palata taaksepäin, mutta seuraava symboli on aina tiedossa. LR-tarkoittaa sitä, että jäsennyspuu voidaan rakentaa alhaalta ylöspäin ilman takaumia tiettyjä jäsennystauluja käyttäen. Yacc:lla voidaan siis esittää BNF:n mukaisia syntakseja (metakieli, jonka Peter Naur esitti vuonna 1960). Tarkastellaan hyvin yksinkertaista Yacc-määrittelyä.

Esimerkin Yacc-lähdekoodi kommentoituna:

```
/* tiedosto /users/int/et/teksti/dokut/esim1.y */

%token IDENTIFIER

%%

expression
        : expression '-' IDENTIFIER
        | IDENTIFIER

%%
```

Token tarkoittaa päätesymbolia (terminal symbol) eli jakamatonta symbolia. Esimerkiksi ohjelmointikielien syntaktiset käskysanat ja numerot ovat päätesymboleja. Vakiintunut tapa on se, että päätesymbolit merkitään Yacc:ssa isoilla kirjaimilla, kun taas johdetut symbolit (non-terminal symbols) kirjoitetaan pienillä kirjaimilla. Esimerkissä on vain yksi päätesymboli, IDENTIFIER. Expression on johdettu symboli, ja se voi rekursiivisesti sisältää toisen expression-symbolin. Rekursio on tyypillistä Yacc-syntakseille, kuten tästäkin esimerkistä käy ilmi.

Edellä kuvattu esimerkki on täydellinen ja laillinen Yacc-syntaksin mukainen ohjelma. Tosin sen suorittava osa puuttuu, joten esimerkin syntaksi ei aikaansaa minkäänlaisia toimintoja. Jäsennyspuu kuitenkin voidaan generoida, ja tilajäsennyskuvaus on seuraava:

```
state 0
 $accept : _expression $end

 IDENTIFIER  shift 2
 .  error

 expression  goto 1

state 1
 $accept :  expression_$end
 expression :  expression_- IDENTIFIER

 $end  accept
 -  shift 3
 .  error


state 2
 expression :  IDENTIFIER_    (2)

 .  reduce 2
```

```
state 3
 expression :  expression -_IDENTIFIER

 IDENTIFIER  shift 4
 .  error


state 4
 expression :  expression - IDENTIFIER_     (1)

 .  reduce 1
```

Yacc lisää kielioppiin alkutilan (state 0), ja asettaa jäsennyksen asemamerkin (position marker) ensimmäisen jäsennettävän symbolin (non-terminal symbol) edelle. Asemamerkki asetetaan aina kaikkien mahdollisten jäsennettävien symbolien eteen (kaikkiin kysymykseen tuleviin lauseisiin). Näin syntyvien jäsennysmahdollisuuksien joukkoa kutsutaan tilaksi (state).

Tilajäsennys etenee seuraavasti: jokaisella tilan osalla (jokaisella mahdollisella jäsennettävällä symbolilla) asemamerkki siirretään jäsennettävän symbolin yli. Näin syntyy uusia tiloja, jotka jäsennetään vastaavalla tavalla. Jäsennys ei voi jatkua loputtomiin, koska sääntöjä, jäsennettäviä symboleja ja päätesymboleja on äärellinen määrä.

Ajatellaan alkutilaa, jossa expression-symbolia edeltää asemamerkki. Koska expression ei ole päätesymboli, se korvataan kaikilla mahdollisilla expression-rakenteilla. Kun asemamerkki siirretään hajautetun expression-symbolin taakse, päästään tilaan 1. Vastaavsti päästään tilaan 2, kun asemamerkki siirretään IDENTIFIER-symbolin taakse. Tilassa 2 ratkaisu on yksinkertainen: koska kyseessä on päätesymboli, jäsennys päättyy, koska jäsennyksen tämä haara voidaan pelkistää (reduce) peruselementiksi eli päätesymboliksi.

Yacc:ssa päätesymboleja ovat paitsi token-käskyllä määritellyt symbolit myös ne yksittäiset merkit, jotka esiintyvät syntaksiesityksessä. Näin ollen tilasta 1 edetään tilaan 3 siten, että asemamerkki siirretään '-'-merkin oikealle puolelle. Tilassa 3 tilalaajennusta ei enää voida jatkaa, joten tilassa 4 jäsennyksen tämäkin haara voidaan pelkistää. Pelkistyksen vaihtoehto on siirto (shift), joka tarkoittaa siirtymistä tilasta toiseen (tai asemamerkin siirtämistä jäsennettävän symbolin yli).

Jäsentäminen ei aina ole ongelmatonta, vaan jotkut kieliopit eivät edellä esitetyn algoritmin avulla pelkisty yksikäsitteisesti. Ristiriitatilanteita (conflicts) on kahta päätyyppiä: siirto-pelkistys ja pelkistys-pelkistys -ristiriitoja.

Siirto-pelkistys -ristiriita ei yleensä aikaansaa vakavia ongelmia, vaan usein jäsennys voidaan ristiriitatilanteesta huolimatta ratkaista yksikäsitteisesti. SP-konflikti tarkoittaa sitä, että algoritmi voisi rat-

kaista tilanteen yhtä hyvin kahdella eri tavalla: joko siirtää siirtää seuraavan jäsennyshaarakkeen uudeksi tilaksi tai pelkistää jäsennyshaaran. Hyvä esimerkki on if-then-else -lause. Kun jäsennys on käsitellyt if-then -osan, on epäselvää, jäsentääkö lause if-then vai if-then-else -rakenteen mukaisesti (else-osa ei yleensä ole pakollinen osa lausesyntaksia). Ongelma ratkeaa tässä tapauksessa lähes itsestään, sillä Yacc ratkaisee SP-ongelman valitsemalla siirron. Tässä tapauksessa (kuten useimmissa tapauksissa) ratkaisu on oikea: jos else-osa on mukana lauseessa, se on jäsennettävä, ja jos se ei ole mukana, pelkistetään lause if-then -osan jälkeen.

Pelkistyt-pelkistys -konflikti (reduce-reduce) on yleensä oire kieliopin vakavasta ristiriitaisuudesta. Tällöin jäsennysalgoritmi voisi yhtä hyvin pelkistää lauseen kahden eri säännön mukaisesti. Vaikka Yacc ratkaisee ongelman oletusarvonsa mukaisesti (pelkistää aikaisemman säännön mukaisesti), PP-konflikti saattaa olla arvaamaton ja vaikeasti hallittava. Vakavasti otettava suositus onkin se, että PP-konflikteja ei tule sallia, vaan ne tulee poistaa kielioppia korjaamalla.

Vaikka Yacc-koodi on esimerkissä vain muutaman rivin mittainen, siitä generoituu yli 10 kilotavun suuruinen C-lähdekoodi. Tämä paljastaa yacc:n tapaisen ohjelman tarpeen: taulukkoviittauksiin perustuvat jäsentäjät edellyttävät niin suurta ohjelmointityötä, että ne jäisivät monessa tapauksessa tekemättä, mikäli Yacc:n tyyppistä koodigeneraattoria ei olisi. Taulukkoviittauksiin perustuvat jäsennystapa on nopea ja periaatteiltaan selkeä. Vastaavan jäsentäjän pystyy todennäköisesti koodaamaan puhtaasti C:llä huomattavasti pienemmällä koodimäärällä, mutta haittapuolena olisi todennäköisesti suhteellinen tehottomuus ja koodin epäselvyys.

12.2 MAT-TULKIN SYNTAKSI JA KOODIN SUORITUS

Edellisessä kappaleessa esiteltiin lyhyesti Yacc-työkalun käyttöä ja toimintaperiaatteita. Tässä kappaleessa esitellään Matin kielioppia ja sen Yacc-esitystä. Koodin perusrunko on omaksuttu C-gurujen Kernighanin ja Piken kirjasta: "The unix programming enviroment". Kirjoittajat eivät ole pelkästään hyödyntäneet Yacc:ia tehokkaasti, vaan he ovat rakentaneet tehokkaan tulkkikoneen, josta on helppo omaksua ajatuksia ja ratkaisutapoja moneen erilaiseen käytännön ohjelmointiongelmaan.

Tulkkikone eroaa tavanomaisesta "on fly" -tyyppisestä tulkista olennaisesti. On fly -tulkissa lauseet suoritetaan saman tien, jolloin heti tulkkauksen jälkeen kutsutaan tulkattua lausetta vastaavaa toimintoa. Tämä on yksinkertainen ja ohjelmoijalle helppo tapa, mutta useissa sovelluksisa se on myös tehoton ja hidas tapa. Jokainen lause joudutaan selaamaan, jäsentämään ja tulkkaamaan kaikilla suorituskerroilla, ja esimerkiksi for- tai while-luuppien suoritus on kankeaa.

Pinokonetulkki on ohjelma, joka kokoaa suoritettavia lohkoja pinoon ko-

neeksi kutsuttuun osoitinvektoriin. Kun kokonainen lohko on pinokoneessa, sitä voidaan suorittaa useita kertoja yhdellä tulkkauskerralla. Muuttujien arvot saattavat tietenkin muuttua, mutta niitä voidaan päivittää toisen keskeisen elementin, aineistopinon, avulla.

Matissa on Kernighanin ja Piken esimerkin mukaisesti päädytty tällaiseen ratkaisuun. Yacc-syntaksin mukaisia toimintoja generoidaan koneeseen, ja kun koko lohko on koneessa, sitä voidaan suorittaa niin kauan kuin sen suorittamiseksi tarvittava ehto on voimassa. Matissa jokainen syntaksin lause sisältää oman suorittavan aliohjelman, jonka mukaan koodia suoritetaan. Koneeseen laitetaan aliohjelmiin osoittavat pointterit, joiden mukaan suoritusta ohjataan. Mat-tulkki vastaa itse asiassa periaatteiltaan yksinkertaista pinologiikalla varustettua tietokonetta: kun syötteestä löydetään operandi, generoidaan koodi joka laittaa sen pinoon. Esimerkiksi yksinkertaisen sijoituslauseen tulkkaus tehdään koneperiaatteella seuraavasti (Kernighan-Pike, 1984):

Lause: x = 2 * y

Operaatio	Selitys
constpush	- Vakio pinoon, vakion arvo on...
2	- 2
varpush	- Muuttuja pinoon, muuttuja on...
y	- y
eval	- Evaluoidaan muuttujasymboli y. Se etsitään symbolitaulusta ja symboliosoitin korvataan muuttujan arvolla (tämä on helppoa C-kielen union-käsitteen avulla)
mul	- Vakio 2 ja muuttujasta evaluoitu vakio (arvoltaan y:n arvo) kerrotaan keskenään. Tulos jää pinoon
varpush	- Muuttuja pinoon, muuttuja on...
x	- x
assign	- sijoitetaan tulos muuttujaan x pinoon jää pinnalle x
pop	- puhdistetaan pino
STOP	- suoritus loppuu

Pienen operaation suorittaminen vaatii monia osasuoritteita, mutta logiikka sinänsä on yksinkertainen.

Seuraavassa on listattu Matin Yacc-syntaksia. Listausta on osin lyhennetty, mutta kaikki olennaiset osat on käyty läpi. Listaukseen on lisätty kommentteja, ja ne on tarkoitettu luettavaksi välittömänä dokumenttitekstin jatkeena. Koska kommentoitu listaus on havainnollisempi kuin pelkkä verbaalinen kuvaus, mat-syntaksin kuvaus rakentuu kommentoidun listauksen varaan. Periaate on se, että jokaista koodin palasta edeltää selittävä kommentti.

```
/**********     mat.y / MAT ************/

/* Yacc-tiedosto on .y-loppuinen. Tiedoston alussa on include- ja makro-
määrittelyt. Tämän jälkeen määritellään unioni, joka sisältää kaikki ne
tyypit, joita jäsentäjän eri symboleilla voi olla. Tämä unioni on näppä-
rä silloin, kun siirretään elementtejä selaajasta jäsentäjälle. */


%{

#include "mat.h"
#define code2( c1, c2 )         code( c1 ); code( c2 )
#define code3( c1, c2, c3 )     code( c1 ); code( c2 ); code( c3 )

%}
%union {
        Symbol *sym;    /* symbolitauluosoitin */
        Inst   *inst;   /*  konekäsky  */
        int     narg;   /*  funktioargumenttien lukumäärä  */
        int     def;    /* apuliput, esim. lauseen loppumerkki */
}


/* Päätesymbolit: jokaisella päätesymbolilla on oma tyyppinsä, jonka
niitä käsittelevät aliohjelmat palauttavat. Mahdolliset tyypit ovat ne,
jotka on määritelty tyyppiunionissa %union. */


%token <sym>  NUMBER CONST VAR DIM VEC MAT QUE UNDEF FUNC PROC
              .                          .
              .                          .
%token <def>  EOST EOBL CONTINUE COMBEG COMEND


/* Ei-päättävillä symboleilla on myös omat tyyppinsä, jotka luetellaan
%type-komennolla.  Mahdolliset tyypit ovat samoja kuin %token-symboleil-
lakin. */


%type  <inst> cond while if for begin end graph elem
              .                 .
              .                 .
%type  <narg> arglist vmexl


/* Right- ja left-komennoilla määrätän symbolien assosiatiivisuutta.
Tämä tarkoittaa sitä, että siirretäänkö lauseen loppuosaa pinoon vai
pelkistetäänkö jo jäsennetty lauseke (vrt. SP-konflikti). Matemaattisis-
sa lausekkeissa on tyypillistä, että sijoitustoiminnot ovat oikealle as-
sosiatiivisia, kun taas operaattoritoiminnot ovat vasemmalle assosiatii-
visia.  Esimerkiksi lauseke 3-4+7 voidaan jäsentää kahdella eri tavalla.
```

Jos jäsennys on oikealle assosiatiivista, aluksi lasketaan 4+7=11, ja tämän jälkeen 3-11= -8. Tämähän on yleisen periaatteen mukaan väärin. Jos lauseke taas jäsennetään vasemmalla assosiatiivisuudella, ensin lasketaan 3-4= -1 ja sitten -1+7=6. Länsimaisen merkintätavan mukaan vasen assosioituvuus on oikein. Muuttujien presedenssiä voidaan säädellä määrittelyosassa. Mitä korkeampi presedenssi, sitä myöhemmin symbolit esitellään left-right-nonassoc -listassa. Näin ollen esimerkiksi sijoitussymboli '=', jonka presedenssi on alhaisin, on listassa ensimmäisenä. Nonassoc-merkintä tarkoittaa sitä, että symboli ei assosioidu puoleen eikä toiseen. */

```
%right '=' PADD PSUB PMUL PDIV
    .                .
    .                .
%left  GT GE LT LE EQ NE
```

/* Äskeiset olivat alkumäärittelyjä. Varsinainen kielioppi alkaa tästä. Yacc:ssa varsinainen kielioppi rajoitetaan rivin alussa olevien '%%'-merkkien väliin. */

```
%%
```

/* State 0 generoidaan tästä. Perusrakenne, johon kaikki redusoituu, on list. Se voi olla tyhjää syötettä (white characters), defn-lause, statement, virhetoiminto tai jokin edellisten rekursiivinen yhdistelmä Yyerrok on Yacc:n generoima virherutiini, jonka avulla jäsennyrutiinista poistutaan, kun se on havaittu syntaktisesti laittomaksi. */

```
list:     /*  tyhjä lause  */
        | list EOST
        | list defn    EOST
        | list stmt    EOST { code( STOP ); return 1;  }
        | list error   EOST { yyerrok; }
        ;
```

/* Tarkastellaan seuraavaksi stmt-nimistä jäsennettävää symbolia. Tämä symboli on Matissa keskeisessä asemassa, sillä se sisältää mm. luuppirakenteet, matriisilauseet, muuttujasijoitukset ja regressiokutsun. Matriisilauseet on tosin koottu omaksi mats-nimiseksi tyypikseen, mutta ne pelkistyvät jäsennyksen päätasolle stmt-lauseen kautta. Listausotteesta on poistettu monia stmt:n osia, mutta keskeiset rakenteet on selityksineen listattu alla. */

```
stmt:     mats    /* Stmt voi olla mm. matriisilause */
```

```
/* Asgn-toiminto ei tässä jäsennyksen vaiheessa generoi koneeseen muuta
kuin pop-komennon, jonka tarkoituksena on tyhjentää pino sijoitustoimin-
nan jäljiltä (vrt. esimerkki edellä). Muut generointikomennot sijoite-
taan koneeseen asgn-lauseen jäsennyksen yhteydessä. Kun jäsennys etenee
tähän vaiheeseen, pinoa ei siis puhdisteta heti, vaan koneeseen luodaan
koodi sen puhdistamiseksi. Code-funktio suoritetaan välittömästi, ja tä-
mä laittaa koneen seuraavaan vapaaseen kohtaan (kohtaan, jota käskylas-
kuri osoittaa) osoittimen, joka osoittaa pop-funktioon */

        | asgn                                  { code( pop );     }

/* Regressiokutsussa määritellään selitettävä ja selittävät muuttujat,
havaintojänne ja kommenttilause. Isoilla kirjaimilla merkityt päätesym-
bolit ovat kutsun kiinteitä käskysanoja, jotka eivät välttämättä vastaa
niitä merkkijonoja, jotka käyttäjä antaa tulkille. Vastaavuudet käyttä-
jän antamien komentojen ja Yacc-parserin välillä määritellään ohjelman
käynnistämisen yhteydessä vastaavuustaulukon mukaan. Yleinen periaate on
se, että todelliset käskyt annetaan pienillä kirjaimilla, ja niitä vas-
taavat Yacc-päätesymbolit ovat samoja mutta isokirjaimisia. Tässä sään-
nössä on muutamia pieniä poikkeuksia. Kun REG-lausetta jäsennetään,
välitön seuraus on code2:n suoritus (joka suorittaa code-funktion kah-
delle funktiolle). Mielenkiintoista on se, että koneeseen ei laiteta ai-
noastaan funktiokutsuosoittimia, vaan myös parametreja voidaan välittää
koneen kautta. Regressiokutsussa ongelmana on se, että selittävien
muuttujien lukumäärä on tuntematon. Tästä syystä koneelle on välitettävä
tieto siitä, kuinka monta selittäjää pinosta on poimittava regression
havaintomatriisiin. Tieto välitetään siten, että Inst-konvertoitu selit-
täjien lukumäärä laitetaan koneeseen funktio-osoittimen asemesta. $6 on
tämä lukumäärä, ja se viittaa jäsennettävän lauseen kuudennen symbolin
palauttamaan arvoon eli symbolin vmexl palauttamaan arvoon. Vmexl:n
tyyppi on määritelty edellä ja itse vmexl myöhemmin tässä kappaleessa.
Yacc-syntaksissa voidaan siis helposti viitata eri symbolien palautta-
miin arvoihin. Varsinainen regressiokoodi ei kuulu Mattiin. Mat ainoas-
taan kokoaa havaintomatriisin väliaikaiseen tiedostoon ja käynnistää
regressio-ohjelman lapsiprosessikseen. Tällainen ratkaisu hidastaa jon-
kin verran regressiokutsun toteutusta, mutta toisaalta itse Mat-ohjelman
koko on jonkin verran pienempi, kun regressio on eriytetty omaksi ohjel-
makseen */

        | REG DEP relis ',' IND vmexl ',' SPAN expr TO expr ',' REM sexpr
              { code2( freg, ( Inst )$6 );     }

/* Matissa funktio palauttaa aina arvon (vrt. Pascal-ohjelmointikieli),
ja arvon palauttaminen on pakollista. Mikäli haluaa käyttää aliohjel-
maa, joka ei palauta mitään, on käytettävä procedure-määrittelyä vastaa-
vasti kuin Pascalissa. defnonly-funktio valvoo välittömästi (koneen ul-
kopuolella), että return-käskysanaa käytetään proseduuri- tai funktioää-
```

rittelyn sisäpuolella. Niiden ulkopuolella tämä käskysana on laiton. Procret-funktio generoi koodin, joka ylläpitää funktiokutsupinoa ja huolehtii siitä, että ohjelman suoritus jatkuu oikeasta kohdasta funktion loputtua. Procret siis etsii kutsupinosta paluuosoitteen eli pääohjelmatason.

```
| RETURN { defnonly( "return" ); code( procret ); }
```

/* Proseduurikutsu käsittää aliohjelmatunnuksen, jonka perässä on sulkeet. Mikäli proseduuriin välitetään argumentteja, ne annetaan arglist:ssa sulkeiden sisällä. Koneeseen laitetaan hyppyfunktio call, kutsuttavan proseduurin tunnus ja funktion argumenttien lukumäärä. Jos symbolin halutaan palauttavan jotakin, se osoitetaan tunnukseen '$$'. Tässä tapauksessa jäsennettävä symboli palauttaa begin-symbolin palauttaman arvon. Begin palauttaa osoitteen seuraavaan vapaaseen koneen osoitinpaikkaan, johon koodia voidaa generoida. Proseduurit määritellään erillisellä lauseellaan, joka määritellään 0-tilan list tasolla. */

```
| PROCEDURE begin '(' arglist ')'
        { $$ = $2; code3( call, ( Inst )$1, ( Inst )$4 ); }
```

/* Print ja input-toiminnot ovat Basic-ohjelmointikielen kaltaisia. molemmille voi antaa rekursiivisen listan, joka sisältää mielivaltaisen määrän numeerisia skalaarimuuttujia tai merkkijonomuuttujia. Print tulostaa listan päätteelle, kun taas input lukee muuttujien arvoja päätteeltä. Jos haluaa tehdä interaktiivisia sovelluksia, input on hyödyllinen valintastruktuuri. */

```
| PRINT prlist          { $$ = $2;}
| INPUT inputlst        { $$ = $2;}
```

/* While-silmukka sisältää while-symbolin, lopetusehdon ja alku- sekä loppuosoitteet (begin ja end), joiden välissä on varsinainen suoritettava osa eli stmt (statement). Koska myös while-käsky on osa stmt-syntaksia, rekursiivinen määrittely eli sisäkkäiset while-silmukat ovat mahdollisia. stmt:n yhteydessä oleva while ei ole while-käskysana (se on pienillä kirjaimilla), vaan WHILE-päätesymbolin yhteydessä generoidaan ns. while-koodi, ja while (pienellä) pitää sisällään osoittimen tähän koodiin sekä osoittimet kahteen tyhjään konekäskypaikkaan. Näihin paikoihin sijoitetaan luupin alku- ja loppuosoitteet tämän stmt-syntaksin yhteydessä. End-syntaksi sisältää paitsi loppuosoitteen etsimisen myös suorituksen loppukoodin generoinnin. Execute-niminen funktio suorittaa koneessa olevia käskyjä läpi niin kauan kuin koneosoittimen arvo poikkeaa NULL-osoittimesta. Tyhjä osoitin on Matissa STOP-merkki, ja kun esimerkiksi while-koodia generoitaessa halutaan varata koneesta kaksi

```
paikkaa myöhempää käyttöä varten, tämä käy käskyllä code2( STOP, STOP ).
Kun while-silmukka loppuu, end-sisältää paluuosoitteen eli seuraavan
suoritettavan konekäskyn.  */


   | while cond begin stmt end
       { $1[1] = ( Inst )$3;     /* luupin runko-osa */
         $1[2] = ( Inst )$5; }  /*  paluuosoite, jos ehto on epätosi  */
   ;


/* Kun WHILE-päätesymboli löydetään syötteestä, koneeseen generoidaan
välittömästi whilecode-funktio ja kaksi tyhjää paikkaa osoittamaan luu-
pin alkua ja luupin jälkeistä seuraavaa käskyä */


while:    WHILE { $$ = code3( whilecode, STOP, STOP ); }
          ;


/* Luupit ja if-lause tarvitsevat lohkolleen toteutusehdon. Tämä ehto on
yksinkertaisesti numeerinen expression, joka on epätosi arvolla 0 ja
muilla arvoilla tosi. Expr:n määrittely esiintyy koodissa myöhemmin,
mutta lyhyesti sanottuna se tarkoittaa mitä tahansa numeerista skalaari-
lauseketta, esimerkiksi 2+4+6/2.3 on expr. */


cond:     '(' expr ')'  { code( STOP ); $$ = $2; }
          ;


/* Kun käsitellään symbolitaulun elementtiä, se on aluksi laitettava pi-
noon. Pinosta symboli voidaan poimia varsinaisen suorittavan funktion
käyttöön pop-funktiolla. Jos halutaan, että Mat-laittaa pinoon symbolin,
aluksi on generoitava koodi symbolin painamiseksi pinoon. Lisäksi konee-
seen on laitettava pinoon asetettavan symbolin osoite. Nämä voidaan suo-
rittaa matex-syntaksin avulla. Aina kun koodista löytyy matriisi, gene-
roidaan koodi sen laittamiseksi pinoon. Ylemmän tason syntaksit voivat
käyttää suoraan matex-symbolia, ja tällöin ei ylemmillä syntaksiasteilla
enää tarvitse huolehtia matriisin pinoamisesta */


matex:    MAT  { $$ = code2( sympush, ( Inst )$1); }
          ;


/* Matriisilause mats (matrix statement) koostuu pitkästä listasta, jos-
sa kerrotaan kaikki mahdolliset matriisilausesyntaksit. Matriisisyntak-
sit eivät ole rekursiivisia, koska rekursiivisten lausekkeiden jäsentä-
misessä tarvitaan aputilaa. Koska matriisit ovat parhaimmillaan hyvinkin
suuria, rekursiiviset matriisilauseet saattavat olla koko järjestelmän
```

kannalta epäedullisia. Tästä syytä Matissa on päädytty ei-rekursiivisiin matriisilauseisiin. Valitettavasti käyttäjäkoodin joustavuus kärsii tästä jonkin verran, koska esimerkiksi pitkät matriisilausekeketjut eivät ole mahdollisia. Mats-syntaksi hyödyntää koneeseen laitettuja pinottuja matriiseja eli matex-symboleja. Koodi on matriisilauseiden osalta helppolukuista, joten sitä on myös miellyttävää päivittää ja muuttaa. Mats sisältää paitsi varsinaisia matriisilauseita myös ns. epäsuoria sarakeviittauksia, joissa käsitellään matriisin yhtä saraketta (muuttujaa). Epäsuoralle sarakeviittaukselle on olemassa oma tietorakenteensa, jossa osoitetaan suoraan matriisin sarakkeeseen. Jokainen epäsuora viittaus on omana elementtinään symbolitaulussa. Koska kaikkiin sarakkeisiin ei yleensä viitata, epäsuora symboli luodaan symbolitauluun vasta, kun sitä ensi kerran kutsutaan. Kun epäsuoria viittauksia (vmex) käytetään, ne on jo generoitu pinoon aivan kuten matriisitkin. */

```
mats:     matex '=' matex '+' matex      { code( matplu );   }
        | vmex  '=' matex expr ROW       { code( rowtovm );  }
        | vmex  '=' matex expr COL       { code( coltovm );  }
        | matex '=' matex '-' matex      { code( matmin );   }
        | matex '=' matex '*' matex      { code( matmul );   }
        | matex '=' matex '/' matex      { code( matdiv );   }
        | matex '=' matex '?' matex      { code( elmul );    }
        | matex '=' vmex                 { code( matisvm );  }
          .          .
          .          .
        | vmex  '=' expr '*' vmex        { code( ekvm );     }
        | vmex  '=' expr '/' vmex        { code( ejvm );     }
        | CHANGE    vmex ',' vmex        { code( changevm ); }
        | vmex  '=' vmex                 { code( vmisvm );   }
        | matex '=' matex                { code( matismat ); }
        | NAME  PRINT  matex             { code( nameprint );}
        | PRINT matex WITH strvex ',' expr ',' sidfr ',' sidfr
                                         { code( niceprint );}
        | matex '=' DIAG  expr           { code( diagset );  }
```

/* Jotkut matriisilauseet on helppo toteuttaa funktioina. Matissa matriisin kääntäminen ja transponointi ovat funktioita, joihin osoitetaan suoraan. Tämä tarkoittaa sitä, että päätesymboli MATF on funktio, joka voidaan asettaa koneeseen. On huomattava, että MATF on oma symbolityyppinsä, ja sillä voi olla useita eri ilmentymiä. MATF voi siis kuvata useaa eri funktiota, kuten esimerkiksi matriisin inverssiä ja transpoosia. Ohjelmaa käynnistettäessä kaikki funktio-osoittimet pannaan symbolitauluun ohjelmakoodissa olevan listan mukaisesti. MATF:n kaltaisia funktioita kutsutaan built in -funktioiksi */

```
        | matex '=' MATF  '(' matex ')'
            {  code2( matf, ( Inst )$3->u.mptr );  }
```

/* Expr-symboli on numeerinen lauseke. Se voi olla mm. sijoituslause (asgn), numero, skalaarimuuttuja, built in -funktio (esimerkiksi C-kirjaston matemaattiset funktiot) tai käyttäjän määrittelemä arvon palauttava funktio. Expr:ssa rekursio on sallittu: näin ollen voidaan jäsentää mielivaltaisen pitkiä numeerisia lausekkeita, joissa myös sulut toimivat normaalien sääntöjen mukaisesti. Loogiset vertailut ovat myös expr-symboleita, koska ne palauttavat aina arvon. Jos looginen ehto on tosi, palautetaan arvo 1, muutoin palautetaan 0. Eräs jäsennys- ja merkintäongelma seuraa unary-merkeistä (etu-miinus ja etu-plus). Esimerkiksi lausekkeessa -3*5 '-'-merkin presedenssin tulisi olla '*'-merkin presedenssiä korkeampi, koska kyseessä ei ole varsinainen lauseke vaan osa yksittäisen luvun määrittelyä. Koska yleensä '-'-merkin presedenssi on matalampi kuin '*'-merkin, sen presedenssi on myös määritelty matalammaksi. Yacc:ssa presedenssiä voidaan vaihtaa %prec-käskyllä. Tällöin samalle merkille voidaan määritellä kaksi päätesymbolia ('-' = UMINUS). */

```
expr:     asgn
        | NUMBER { $$ = code2( constpush, ( Inst )$1 ); }
        | ARG    { defnonly( "%" ); $$ = code2( arg, ( Inst )$1 ); }
        | FUNCTION begin '(' arglist ')'
                 { $$ = $2; code3( call, ( Inst )$1, ( Inst )$4 ); }
        | BLTIN '(' expr ')'
                 { $$ = $3; code2( bltin, ( Inst )$1->u.ptr ); }
        | '(' expr ')'    { $$ = $2; }
        | expr '+' expr { code( add ); }
        | expr '-' expr { code( sub ); }
        | expr '*' expr { code( mul ); }
        | expr '/' expr { code( div ); }
        | '-' expr %prec UMINUS { $$ = $2; code( negate ); }
        | '+' expr %prec UPLUS  { $$ = $2; }
          .          .
          .          .
        | expr  GT expr   { code( gt ); }
        | expr  GE expr   { code( ge ); }
        | expr MOD expr { code( mod ); }
        | expr '^' expr { code( power ); }
        ;
          .          .
          .          .
          .          .
%%
```

12.3 **MATIN TIETORAKENTEET**

Matissa muuttujat ja muut symbolit hajautetaan hash-funktion avulla symbolitauluun, josta ne nopeasti haettavissa. Hash-funktiosta saadut kokemukset ovat olleet hyviä, eikä symbolien hakuaika ole ollut Matin ongel-

mana. Symbolitaulun elementin tietorakenteena on monitasoinen struktuuri, johon voidaan sijoittaan kaikki tarvittavat symbolityypit skalaarimuuttujast matriisiin. Alla on listatttu ja selitetty symbolitauluelementtirakenteen keskeisiä osia:

/* linkitetty lista regressioanalyysin havaintomatriisille, joka välitetään regressiolohkolle vfork-prosessihajautusfunktion avulla. */

```
struct reglist
       { char    *sym;
         char    *var;
         int     cnt;
         double  *data;
         struct  reglist *next;
       };
typedef struct reglist Reglist;

/*  Matin tärkein tietorakenne, symbolitauluelementti.  */

struct symbol                /* Symbolitaulun elementti   */
       { char   *name;              /* nimi */
         short  type;               /* tyyppi */
         union {double val;            /* numeerinen vakio */
                struct { int n;
                         double *adr;
                       } vec;        /* vektori */
                struct { int    c;   /* sarakkeen paikka matriisissa */
                         struct symbol *adr; /* osoitettava matriisi */
                       } vm;  /*  epäsuora sarakeviittaus  */
                struct { int      m, n;   /* dimensiot     */
                         char   **rn;     /* riivnimet */
                         char   **cn;     /* sarakenimet */
                         char    *legend; /* kommentti tms. */
                         double **adr;    /* varsinainen data */
                       } mat;  /* matriisi */
                struct { int n;       /* merkkijono */
                         char *adr;
                       } str;
                struct { int m, n;     /* merkkijonovektori */
                         char **adr;
                       } Str;
                double (*ptr)();          /* C-kirjastofunktio */
                struct symbol *(*vptr)(); /*  epäsuoran viittauksen
                                              funktio  */
                struct symbol *(*mptr)(); /* matriisifunktio */
                int    (*defn)(); /*  Mat-käyttäjän oma
                                      funktio  */
               } u;
         struct symbol *next;
```

```
/* Jos hash-funktio aikaansaa konfliktin kahden symbolin välille, symbo-
lit  linkitetään peräkkäin symbolitaulun samaan kohtaan. Ratkaisu on yk-
sinkertainen ja ehkä jossain määrin tehotonkin, mutta koska hash-funktio
on  hyvä ja jos symbolitaulu allokoidaan riittävän suureksi, lineaarinen
haku yhteentörmäystilanteissa ei käytännössä aiheuta ongelmia. */


      };
typedef struct symbol Symbol;   /*   Symbol-tyypin nimeäminen  */


typedef union Datum  { /*  Data-pinollekin on oma tyyppinsä  */
        double  val;
        double *adr;
        struct { int n; char *adr;} str;
        Symbol *sym;
} Datum;

typedef int  ( *Inst )();  /*  Konekielisen käskyn tyyppi Inst on
                               osoitin, joka osoittaa int-tyypin
                               palauttavaan funktioon  */

/*  Seuraavassa on esitetty koneeseen ja pinoon
   liittyviä tietorakenteita  */


#define NSTACK 256          /* pinon koko */


static Datum stack[NSTACK]; /* itse pino */
static Datum *stackp;       /* osoitin, joka kertoo seuraavan vapaan
                               pino-osoitteen */

#define NPROG  25000        /* koneen koko, 25000 Inst-tyyppistä
                               konekielistä käskyä */


Inst    prog[NPROG];        /* itse kone */
Inst   *progp;              /*  seuraava vapaa
                               konekäskyosoitin  */
Inst   *pc;                 /*  suoritusaikainen
                               käskylaskin  */
Inst   *progbase = prog;    /* aliohjelman alku */
int     returning;          /*  saa arvon yksi, jos aliohjelmasta
                               palataan päätasolle  */

typedef struct Frame {      /*  rakenne, joka pitää kirjaa
                               käyttäjän määrittelemistä aliohjelmista
                               ja niiden kutsuista  */
        Symbol *sp;         /* funktio- tai proseduuriosoitin */
        Inst   *retpc;      /* paluuosoite */
```

```
        Datum  *argn;       /* funktion argumentit */
        int     nargs;      /* argumenttien lukum{{r{ */
} Frame;


#define NFRAME 100

Frame frame[NFRAME];        /* itse funktiopino */
Frame *fp;                  /* funktiopino-osoitin */
```

Tässä kappaleessa on tyydytty lähes ainoastaan luettelemaan tietorakenteita. Niiden varsinaisesta käytöstä ei juuri ole ollut mainintaa, mutta siihen paneudutaan vasta kappaleessa, jossa esitellään tärkeimpiä ohjelman funktioita ja toimintoja.

12.4 KESKEISIÄ FUNKTIOITA

Alla on kuvattu koodina joitakin ohjelman toiminnan ymmärtämisen kannalta keskeisiä funktioita. Koko listausta ei ole kommentoitu ja esitettyjäkin koodeja on osin supistettu kuvauksen selventämisen parantamiseksi. Kaikkiaan Matissa on yli 5000 riviä lähdekoodia (noin 140 kilotavua), joten sen täydellinen ja seikkaperäinen kuvaaminen ei varmaankaan vastaa tarkoitustaan.

```
/* Pääohjelma on yksinkertainen. Aluksi asetetaan setjmp-funktio, jossa
määritellään virherutiineiden paluukohdat. Tämän jälkeen määritellään
kaksi yleisintä virhesignaalia poimittavaksi, jos ne aikaansaadaan Matin
suorituksen aikana. Jos virhesignaalikeskeytykset toteutuvat, siirrytään
nimenomaisiin virherutiinifunktioihin segcatch ja fpecatch. Näiden avulla Mat toipuu käytännössä katsoen kaikista käyttäjän ja ohjelman tekijän
virheistä. Yleensä esimerkiksi SIGSEGV (segmentation violation) johtaa
prosessin välittömään kaatumiseen, mutta jos signaali käsitellään oikein
ja kone voidaan palauttaa alkutilaan, mitään vaaraa ei yleensä aiheudu.
Joissakin hyvin harvinaisissa tilanteissa konetta ei saada alkutilaan,
jolloin ohjelma on keskeytettävä. Tässä vaihessaa etsitään käyttäjän kotihakemistosta '.matrc'-niminen tiedosto, jonka alkuasetukset aktivoidaan startup-funktiolla. init-funktio asettaa symbolitauluun päätesymbolit, sisäänrakennetut funktiot ja joitakin vakioita. Varsinainen suoritus on päätasolla yksinkertainen: niin kauan kun syötettä tulee niin
suorita se. Syöte loppuu tiedoston loppumerkkiin. Mikäli tiedoston jälkeen syötteestä ei löydy uutta tiedostoa, ohjelma katkeaa. */


main( argc, argv )
char *argv [];
{ int i;
  setjmp( begin );
  signal( SIGSEGV, segcatch );
```

```
  signal( SIGFPE,  fpecatch );
  startup( ".matrc" );
  gargv = argv + 1;
  gargc = argc - 1;
  init( );
  while( moreinput( ) ) run( );
  return 0;
}
```

/* Moreinput on rekursiivinen funktio, joka tutkii, löytyykö syötteestä uutta tiedostoa kun edellinen loppuu. Tämä mahdollistaa sen, että Matille voi käynnistyksen yhteydessä antaa parametrina mielivaltaisen määrän suoritettavia tiedostoja. Gargc ja gargv ovat vastaavia kuin saman nimiset muuttujat ilman g-kirjainta. Ainoa ero on se, että g-alkuiset muuttujat ovat globaaleja. */

```
moreinput()                          /*                          */
{
  if( gargc-- <= 0 ) return 0;
  if( fin && fin != stdin ) fclose( fin );
  infile = *gargv++;
  lineno = 1;
  if( strcmp( infile, "-" ) == 0 )
   {
   fin = stdin; infile = 0;
   }
  else if ( ( fin = fopen( infile, "r" ) ) == NULL )
   {
   spr2( "%s: can't open %s0, progname, infile );
   return moreinput( );
   }
  return 1;
}
```

/* Run on eräs Matin perusfunktioista. Aluksi se asettaa virhetilanteen paluuosoitteen uudelleen samoin kuin signaalipoimintakomennotkin. Tämän jälkeen se asettaa koneen alkutilaan, jäsentää syötettä tiedoston loppumerkkiin asti ja suorittaa jäsennyksen tuloksena generoidun koodin. Funktio yyparse on Yacc:n tuottama vakioniminen funktio, joka jäsentää syötteen. */

```
run( )
{ setjmp( begin );
  signal( SIGSEGV, segcatch );
  signal( SIGFPE, fpecatch );
  for( initcode( ); yyparse( ); initcode( ) )
      execute( progbase );
}
```

/* Koneen nollaus on yksinkertainen toimenpide. Käskylaskuri laitetaan osoittamaan koneen alkuun, ja samoin pino-osoitin siirretään pinon alkuun. Tässä vaiheessa sisäkkäisiä funktiokutsujakaan ei ole funktiopinossa, joten sekin voidaan nollata. Kone ei myöskään ole paluutilassa, joten returning-lippu voidaan nollata */

```
initcode( )
{       progp  = progbase;
        stackp = stack;
        fp     = frame;
        returning = 0;
}
```

/* Pinoelementtien laittaminen pinoon ja poistaminen pinosta. Nämä operaatiot voidaan toteuttaa myös makroilla, jolloin koodi nopeutuu, ja mikä yllättävää, myös lyhenee. Tämä on seurausta siitä, että konekielisessä funktiokutsussa (nyt tarkoitetaan aitoa ympäristön konekieltä) tilaa varataan myös parametreille ja paluuosoitteelle (aivan kuten matissakin). Jostakin syystä näiden aiheuttama lisätilanvaraus on suurempi kuin push- ja pop-makrojen. Näin ollen makrojen käytöstä on pelkkää etua. Käytännössä kuitenkin makrojen hallinta on sen verran vaikeampaa, että loppujen lopuksi on päädytty funktioiden käyttöön. */

```
push( d )
Datum d;
{
if( stackp >= &stack[NSTACK] )
 execerror( "stack too deep", (char *) 0 );
 *stackp++ = d;
}

Datum pop()
{
if( stackp == stack )
 execerror( "stack underflow", (char *) 0 );
return *--stackp;
}
```

```
/* Symbolin poimiminen koneesta ja sen siirtäminen pinoon. Samalla kone-
käskyosoitinta siirretään yhdellä eteenpäin */


sympush()
{ Datum d;  d.sym = (Symbol *)(*pc++);  push(d);}


/* Edellisessä kappaleessa kuvattiin, miten osoitin whilecode-funktioon
generoitiin koneeseen. Kun kone etenee suoritusvaiheeseen, while-silmuk-
ka toteutetaan seuraavasti. Whilecode-osoitteen yhteydessä koneeseen
laitettiin kaksi NULL- eli STOP-merkkiä, jotka while-symbolin jäsennyk-
sen yhteydessä täytettiin lohkon alku- ja paluuosoitteilla. Aluksi nämä
osoitteet sivuutetaan, ja silmukan suoritus aloitetaan ehto-osasta
(savepc+2). Luupin cond-osa jättää pinon päälle lukuarvon 1, jos suori-
tusehto on voimassa, muussa tapauksessa pinoon jää arvo 0. Niin kauan
kuin ehto on voimassa eli execute( savepc+2 ) jättää pinon päälle nol-
lasta poikkeavan arvon (tämä tarkistetaan joka kerta kun runko-osa on
suoritettu), käyttäjän antamaa luuppia toistetaan. Varsinaisen suoritet-
tavan osan osoite oli whilecoden jälkeisessä paikassa. tämä osoite voi-
daan sopivasti castattuna välittää execute-funktion avulla suoritetta-
vaksi. Joissakin tilanteissa (esimerkiksi syötteen loppuessa) execute:n
aikana paluulippu saattaa nousta ylös. Tällöin luuppi keskeytetään ja
lähdekoodista poistutaan breakin avulla. Toinen poistumistapa on se,
että suoritusehto tulee epätodeksi. Tällöin luuppi keskeytetään ja suo-
ritusta jatketaan paluuosoitteen kertomasta osoitteesta. */


whilecode()
{ Datum d;
  Inst *savepc = pc;
  execute( savepc + 2 );                    /* ehto-osa */
  d = pop();
  while ( d.val )
    {
    execute( *( (Inst **) ( savepc ) ) ); /* runko-osa */
    if( returning ) break;
     execute( savepc + 2 ); /* ehto-osa */
    d = pop();
    }
 if ( !returning )
    pc = *( ( Inst **) ( savepc + 1 ) ); /* seuraava */
}


/* Call-funktio kutsuu käyttäjän määrittelemää proseduuria tai funktio-
ta.  Nämä kaksi esiintyvät Mat-syntaksissa eri paikoissa: proc on stmt
ja func taas on expr. Kutsufunktio on kuitenkin sama, ja kun syötteessä
esiintyy funktiokutsu, koneeseen generoidaan paitsi call-konekäsky myös
osoitin käyttäjäfunktioon ja argumenttilistaan tai oikeastaan argument-
ti-tyypin palauttamaan arvoon eli argumenttien lukumäärään. Varsinaiset
```

argumentit ovat pinossa, mutta välttämätön tieto niiden lukumäärästä on generoitu koneeseen. Kun suoritusvaiheessa on edetty call-funktioon, execute-suoritusfunktio on jo päivittänyt pc-osoitinta (program counter), joka call-funktion suorituksen aikana osoittaa jo koneen seuraavaan osoitinalkioon, tässä tapauksessa käyttäjäfunktioon. Kaikki käyttäjän määrittelemät symbolit, kuten myös funktiot, laitetaan symbolitauluun, ja näin ollen kyseessä on Symbol-tyyppinen osoitin. Tyyppikonversio varmistetaan (Symbol *)-castilla. Funktio laitetaan aliohjelmapinoon, ja mikäli pino on täynnä, annetaan virheilmoitus ja lopetetaan lohkon suoritus ja palautetaan kone alkutilaan virherutiinifunktion execerror avulla. Funktio-osoittimen jälkeen koneesta löytyy argumenttilista, jota seuraavasta koneosoitteesta funktion jälkeinen ohjelmasuoritus jatkuu. Paluuosoite, funktio-osoitin, argumenttilista ja argumenttien lukumäärä talletetaan funktiopinoon, joka mahdollistaa sisäkkäiset funktiokutsut.

```
call()
{ Symbol *sp = (Symbol *)pc[0]; /* Funktio-osoitin koneesta */
  if( fp++ >= &frame[NFRAME - 1] )
    execerror( sp->name, "call nested too deeply" );
  fp->sp = sp;
  fp->nargs = ( int ) pc[1]; /*  argumenttien lukumäärä  */
  fp->retpc = pc + 2;        /* paluuosoite */
  fp->argn = stackp - 1;     /*  pinon päälimmäinen argumentti  */
  execute( sp->u.defn );     /* funktion suoritus */
  returning = 0;
}
```

```
/* Koodin suoritus. Suoritetaan konepointterien osoittamia funktioita
niin kauan kun osoittimet ovat erisuuria kuin NULL (STOP). Myös return-
ning-lipun aktivoituminen aikaansaa suorituksen lopettamisen. For-luu-
pissa käskylaskuri pc (program counter) laitetaan osoittamaan execute-
funktion argumentin osoittamaan paikkaan. Suoritus aikaansaadaan käskyllä
(*(*pc++))(); Tämä voidaan tulkita seuraavasti: kone on Inst-tyyppi-
nen vektori, ja Inst-tyyppi on osoitin int-tyypin palauttavaan funk-
tioon.  Pc  on  samaa tyyppiä kuin konekin, mutta se osoittaa osoitteeseen, joka puolestaaa osoittaa funktioon. Kyseessä on siis osoittimen
osoitin. Käsky siis suorittaa koneessa olevan osoittimen takaisen funktion, ja kasvattaa laskuria postfix-ilmauksen avulla yhdellä. */
```

```
execute( p )
Inst *p;
{
for(pc = p; !returning && *pc != STOP; )
  (*(*pc++))();
}
```

/* Code-funktio laittaa koneeseen käskyjä ja operandeja. Code palauttaa osoitteen asennetuun käskyyn. Konelaskuria päivitetään siten, että se osoittaa seuraavaan vapaaseen paikkaan. Jos kone täyttyy, siirrytään virherutiiniin. */

```
Inst *code(f)
Inst f;
{ Inst *oprogp = progp;
  if ( progp >= &prog[NPROG] )
    execerror( "program too big", (char *) 0 );
  *progp++ = f;
  return oprogp;
}
```

/* Kuten syntaksikuvauksen yhteydessä jo todettiin, jotkut matriisitoiminnot ovat funktioita, joihin voidaan koodigeneroinnissa osoittaa suoraan. Tämä yksinkertaistaa Yacc-syntaksia ja sallii funktioiden joustavan lisäämisen. Funktioviittaukset on ainakin toistaiseksi rajoitettu matriisien osalta syntakseihin, joissa on yksi lähde- ja yksi tulosmatriisi. Kun selaaja löytää matriisi-tyyppisen päätesymbolin, se palautetaan jäsentäjälle. Jäsentäjä parsii kaikki matriisit matex-symbolin avulla, joka generoi koodin matriisin laittamiseksi pinoon. Kun jäsennyksessä päästään funktiosyntaksiin asti, matriisit ovat jo pinossa. Matriisifunktiot suoritetaan alla olevan aliohjelman avulla seuraavasti. Matriisit "popataan" pinosta, ja koneesta löytyy osoitin suoritettavaan funktioon. Funktio-osoittimen sisältöä *pc castataan sopivasti siten, että kyseessä on itse asiassa funktiokutsu. Monimutkaisen näköinen (*(Symbol *(*)()) (*pc++)) tarkoittaa sitä, että koneosoittimessa on osoitin funktioon, joka palauttaa Symbol-tyyppisen osoittimen. Kyseessä on funktio, joka palauttaa osoittimen Symbol-tyyppiseen osoittimeen. Uloin tähti tarkoittaa sitä, että palautettava symboli siirretään d1:n symboliosaan.

```
matf()
{ Datum d1,d2;
  d2=pop();  d1=pop();
  d1.sym = (*(Symbol *(*)()) (*pc++))( d1.sym,d2.sym );
}
```

/* Esimerkki varsinaisesta syntaksissa esiintyvän matriisilauseen toteutuksesta (joka ei ole funktio-osoitin, vaan koneeseen generoitu koodi). Kahden matriisin elementtien summaus kolmanteen matriisin. Kun koodia suoritetaan, matriisit on jo laitettu pinoon. Aluksi matriisit poimitaan pinosta Datum-elementteihin d1, d2, ja d3. Tämän jälkeen tarkistetaan, onko automaattinen matriisidimensiointi kytketty päälle (autodim-lippu). Jos se on päällä ja selaaja on palauttanut parserille tyhjän matriisin (dimensioiltaan 0,0), matriisin varsinainen data-alue allokoidaan post-

set-funktiolla ensimmäisen syötematriisin d2 dimensioiden mukaisesti. Mikäli matriisit ovat erisuuria ja eqok-funktio palauttaa nollasta poikkeavan arvon, annetaan varoitus. Virherutiiniin ei siirrytä, vaan elementtien summaus tehdään pienimmän matriisin dimensioiden mukaan. Tulosmatriisin rivi- ja sarakeotsakkeiksi sijoitetaan ensimmäisen syötematriisin nimet. Itse summaus onkin alkutoimenpiteiden jälkeen yksinkertainen toimenpide. */

```
matplu()
{
  int   i, j, nr, nc;
  Datum d1,d2,d3;
  d3=pop(); d2=pop(); d1=pop();
  if ( audim && d1.sym->u.mat.m==0 )
    postset( d1.sym, d2.sym->u.mat.m, d2.sym->u.mat.n );
  if ( !eqok( d3,d2,d1 ) )
   warning( "warning: unequal sizes","" );
  namecpy( d1.sym, d2.sym );
  nr = MIN( d1.sym->u.mat.m, d2.sym->u.mat.m );
  nr = MIN( nr, d3.sym->u.mat.m );
  nc = MIN( d1.sym->u.mat.n, d2.sym->u.mat.n );
  nc = MIN( nc, d3.sym->u.mat.n );
  for (i=0; i < nr; i++)
   {
   for (j=0; j < nc; j++)
     d1.sym->u.mat.adr[i][j] =  d2.sym->u.mat.adr[i][j] +
                                d3.sym->u.mat.adr[i][j] ;
   }
 }
```

12.5 SELAAJAN TOTEUTUS

Selaaja on tulkin tai kääntäjän se lohko, joka etsii syötevirrasta syntaksissa esiintyviä symboleita ja välittää ne jäsentäjälle. Selaaja on Matissa yylex-niminen funktio. Sillä on kiinteä nimi, koska Yacc hyödyntää vakionimistä selaajaa. Yylex on Matin monimutkaisin ja suurin yksittäinen funktio. Monimuotoista selaajaa on vaikea tehdä sekä tehokkaaksi että selkeäksi. Yleensä selkeydestä joudutaan tinkimään tehokkuuden kustannuksella, sillä tulkin joutuisan toiminnan kannalta on ensiarvoisen tärkeää, että syötettä pystytään lukemaan niin nopeasti kuin mahdollista. Funktiokutsujen runsas käyttö hidastaa jonkin verran tulkin nopeutta, ja ennen kaikkea tästä syystä niitä ei juuri ole käytetty. Toinen ongelma on se, miten selaaja voidaan järkevästi jakaa osasuoritteisiin. Yksi järkevä tapa on siirtää tehokkuusvastuu valmiille selaajatyökaluille. Unix-ympäristössä tällainen on Lex-niminen selaajageneraattori. Matin esikuvan toteuttajien (Kernighan ja Pike) mielestä puhdas C-koodi on Lexin generoimaa siinä määrin tehokkaampaa, että mainittujen gurujen

valinta oli C-koodi. Matissa lähdettiin samoille linjoille.

Seuraavassa käydään kursorisesti läpi selaajan pääpiirteitä. Koko selaajafunktio on niin pitkä ja hankalalukuinen, että olennaiset piirteet hautautuvat koko koodin alle. Tästä syystä dokumentissa on tyydytty selostamaan vain osaa koodista.

```
yylex()                 /* Mat-selaaja, KRPK 84, HV 87, ET 88 */
{Newline:

/*  Tyhjät merkit (white characters) ovat yhdentekeviä Matille, ne
ohitetaan surutta. Jos spool-tiedosto on auki ja tilassa, jossa myös
komennot syötetään tulostiedostoon, myös tyhjä merkistö välitetään
tuloksiin.  */

  while((c = getc(fin)) == ' ' || c == '')
   { if (sp && !resonly) putc(c, spoout);  }

/* Tiedoston loppuminen katkaisee funktion suorituksen */

  if(c == EOF) return 0;

/*  Numero on eräs päätesymbolityyppi  */

  if(c == '.' || isdigit(c))
  {   double d;
      ungetc(c, fin);
      fscanf(fin, "%lf", &d);
      if(sp && !resonly) fprintf(spoout, " %lg ", d);
      yylval.sym = install("", NUMBER, d);
      return NUMBER;
  }

/*  Käyttäjäfunktioiden argumentit numeroidaan ykkösestä lähtien.
Mikäli argumentin numero ei ole järkevä, annetaan virheilmoitus. Selaaja
välittää parserille tyypin mukaisia arvoja yylval-unionin avulla, joka
on Yacc-koodin alussa määritellyn unionin mukainen. Itse tyyppi palautetaan
yylex-funktion palauttamana arvona. Yacc numeroi päätesymbolit kokonaisluvuilla.  Näille luvuille annetaan päätesymbolimerkintöjen mukaiset vastineet silloin, kun Yacc koodi konvertoidaan C-koodiksi.  */

  if(c == '%')
  {   int n = 0;
      while(isdigit(c = getc(fin)))
           n = 10 * n + c - '0';
      ungetc(c, fin);
      if(n == 0) execerror("strange parameter ref. %...", (char *)0);
      yylval.narg = n;
      if(sp && !resonly) fprintf(spoout, " %d ", n);
      return ARG;
```

```
}
```

/* Jos symboli ei ole numero, se voi olla muuttujatunnus. Jos tunnus alkaa kirjaimella, vakiota kuvaavalla risuaidalla tai alaviivalla, koko tunnus luetaan läpi. Mikäli se on pitempi kuin suurin laillinen tunnuspituus, annetaan virheilmoitus. Jos tunnuksessa esiintyy '.'-merkki, se mahdollisesti on epäsuora sarakeviittaus, mutta tämä ei ole varmaa. Tarkistuslippu nostetaan kuitenkin ylös. */

```
if( isalpha( c ) || c ==  '#'  || c == '_' )
{ Symbol *s;
  int chk_on = 0;
  int type, system = c == '£';
  char sbuf [ VNLEN ], *p = sbuf;
  do { if(p >= sbuf + sizeof( sbuf ) - 1)
       { *p = ' '; execerror( "name too long", sbuf ); }
       *p++ = c;
       if ( c == '.' ) chk_on = 1;  /* Saraketyyppi VM ? */
  } while( ( c = getc( fin )) != EOF &&
       ( isalnum( c ) || c == '_' || c == '.' || c == '\047' ) );
      .                .
      .                .
      .                .
```

/* Kun tunnus on kerätty sbuf-merkkijonoon, voidaan sen olemassaoloa tutkia. Aluksi tarkistetaan, onko se epäsuora sarakeviittaus, jota ei aikaisemmin ole käytetty (eli sellainen, jota vielä ei ole olemassa). Tarkistus tehdään siten, että tunnuksessa esiintyvää pistettä edeltävät merkit poimitaan yhteen vektoriin ja pisteen jälkeiset merkit toiseen. Jos symboli todella on uusi sarakeviittaus, tunnuksen alkuosan mukainen matriisisymboli löytyy symbolitaulusta, ja jälkiosan mukainen sarakenimi löytyy tästä matriisista. Ratkaisu ei kaikilta osin ole onnistunut, sillä sarakenimiä ei hajauteta eikä etsitä hashing-funktion avulla, vaan sitä etsitään otsake otsakkeelta. Tämä ei ole kovin tehokasta, mikäli matriisit ovat suuria. Jos matriisi on olemassa ja siinä esiintyy tunnuksen mukainen sarakenimi, kaikki on kunnossa ja uusi symboli asennetaa install-funktiolla tauluun. Jäsentäjälle palautetaan tyyppi VM */

```
if ( chk_on && !(s=lookup(sbuf)) )
 {
   .        .
   .        .
   .        .
   cc = 0;
   while ( sbuf[ cc ] != '.' )
    {   chk_buf[ cc ] = sbuf[ cc ];
        cc++ ;  }
   chk_buf[ cc ] = '\0';
   cc++ ; /* ohita . */
```

```
    if ( ( s = lookup( chk_buf ) ) && ( s->type == MAT ) )
     { /* VM  löytyi  */
       ck = 0;
       while ( sbuf[ cc ] != '\0' )
        {   chk_buf[ ck ] = sbuf[ cc ];
            ck++ ;  cc++ ;                  }
       chk_buf[ ck ] = ' ';
       cc = 0;
       if ( ( cc = look_for( s, chk_buf ) ) == -1 )
         execerror( "unknown column name : ", chk_buf );
     }
    ss = install( sbuf, VM, 0. );
    ss->u.vm.adr = s;
    ss->u.vm.c   = cc;
    yylval.sym   = ss;
    return ss->type;
  }
```

/* Selaaja ei aina voi suoraan päätellä, mitä tyyppiä käsiteltävä symboli on, vaan usein tutkittavan symbolin tyyppi on riippuvainen varsinaisen tunnuksen jälkeisestä merkistä. Esimerkiksi jos tunnuksen jälkeen esiintyy '('-merkki, tunnus on funktio. Jos merkki taas on avautuva hakasulku, symbolin on matriisi tai vektori. Jos symbolia ei vielä esiinny (eli jos lookup-funktio palauttaa NULLin), ratkaisu on jätettävä jäsentäjälle. Esimerkiksi dim-lausee yhteydessä symbolia ei vielä ole olemassa, vaikka lause muuten olisikin järkevä. Epämääräisen tyypin nimi on QUE. Jos syötteessä esiintyy '$'-merkki, kysymyksessä on merkkijonomuuttuja. Selaajan on tässä vaiheessa tutkittava, onko kyseessä skalaari vai vektori. Switch-lausee default-osiossa palautetaan tavalliset numeeriset skalaarimuuttujat. Johtopäätös siitä, että muuttuja on skalaari, saadaan residuaalina. Jos se ei vielä ole olemassa, se luodaan. Mikäli autodim-lippu on nostettu pystyyn, tämä tarkoittaa sitä, että uuden symbolin oletusarvotyyppi ei ole skalaari, vaan kokonainen matriisi. Tätä ei pidä sekoittaa selaajassa aiemmin esiintyneeseen kohtaan, jossa tunnistetttiin matriisin elementtiä eli itse asiassa skalaaria. */

```
    .           .
    .           .
    .           .
switch (c)   /*  funktio, vektori, skalaari, käskysana ?  */
{ case '(':
    .           .
    .           .
  case '[':    if ( !( s = lookup( sbuf ) ) )
               s = install( sbuf, QUE, 0. );
               yylval.sym = s;
               return s->type;
    .           .
    .           .
```

```
    case '$': /* merkkijonovektori tai merkkijonoskalaari */
              *p++ = ( c = getc( fin ) ); *p = ' ';
              type = ( ( c  = getc( fin ) ) == '[' ) ? NEWSV : NEWSS;
              ungetc( c, fin );
              if ( ( s = lookup( sbuf ) ) == 0 )
                s = install( sbuf, type, 0. );
              yylval.sym = s;
              return s->type;
      .              .
      .              .
    default:      if( !( s = lookup( sbuf ) ) )
                   {
                    if ( !audim ) s = install( sbuf, VAR, 0. );
                    else          s = install( sbuf, MAT, 0. );
                   }
                  yylval.sym = s;
                  return s->type;
  }
      .              .
      .              .
```

/* Selaajan perusperiaatteena on se, että kaikki yksittäisistä merkeistä koostuvat päätesymbolit (esimerkiksi operaattorit '+', '-' jne.) välitetään sellaisinaan parserille. Valitettavasti kaikki operaattorit eivät ole näin yksinkertaisia, vaan kaikkia kaksimerkkiset operaattorit, kuten '+=', '>=' jne. on nimettävä omiksi tyypeiksiin ja tutkittava erikseen. Tähän tehtävään on luotu oma funktionsa (follow), joka tutkii syötteen seuraavan merkin ja palauttaa tähän liittyvän tyypin. Eräs ongelmallinen symboli on kommenttimerkki, joka alkaa '/'-merkillä. Matissa lausee lopumerkki voi olla paitsi ';'-merkki myös rivinvaihto. Tämä on käyttäjän kannalta mukava ratkaisu, sillä interaktiivisen tulkin yhteydessä rivinvaihto on yleisin lauseenlopetusmerkki. Mikään ei silti estä ';'-merkin käyttämistä. */

```
      .              .
      .              .
  switch (c)
  {   case '>' :  return follow( '=', GE, GT );
      case '<' :  return follow( '=', LE, LT );
      .              .
      .              .
      case '!' :  return follow( '=', NE, NOT );
      .              .
      .              .
      case '\n':  lineno++; yylval.def = c; return EOST;
      case ';' :  if ( follow( '\n', EOST    , ';' ) == EOST )
                   lineno++;
                  yylval.def = c; return EOST;
          .              .
```

```
    default  :  return c;
}
```

LÄHTEET

Kernighan, B.W ja Pike, R. — The Unix programming environment Prentice-Hall, New Jersey, 1984.

Wilkinson, Reinsch: — Linear Algebra. Die Grundlehren der matematischen Wissenschaften in Einzeldarstellungen, Band 186, Heidelberg 1971.

ELINKEINOELÄMÄN TUTKIMUSLAITOS (ETLA)
The Research Institute of the Finnish Economy
Lönnrotinkatu 4 B, SF-00120 HELSINKI Puh./Tel. (90) 601 322
Telefax (90) 601 753

KESKUSTELUAIHEITA - DISCUSSION PAPERS ISSN 0781-6847

No 260 ERKKI KOSKELA, Saving, Income Risk and Interest Rate Wedge: A Note. 12.05.1988. 10 pp.

No 261 MARKKU KOTILAINEN, Medium-Term Prospects for the European Economies. 02.06.1988. 45 pp.

No 262 RITVA LUUKKONEN - TIMO TERÄSVIRTA, Testing Linearity of Economic Time Series against Cyclical Asymmetry. 08.06.1988. 30 pp.

No 263 GEORGE F. RAY, Finnish Patenting Activity. 13.06.1988. 19 pp.

No 264 JUSSI KARKO, Tekniikkaerojen mittaaminen taloudellis-funktionaalisen ja deskriptiivisen indeksiteorian puitteissa. 28.06.1988. 57 s.

No 265 TIMO SAALASTI, Hintakilpailukyky ja markkinaosuudet Suomen tehdasteollisuudessa. 01.08.1988. 75 s.

No 266 PEKKA ILMAKUNNAS, Yritysaineiston käyttömahdollisuuksista tutkimuksessa. 18.08.1988. 40 s.

No 267 JUSSI RAUMOLIN, Restructuring and Internationalization of the Forest, Mining and Related Engineering Industries in Finland. 19.08.1988. 86 pp.

No 268 KANNIAINEN VESA, Erfarenheter om styrning av investeringar i Finland. 26.08.1988. 17 s.

No 269 JUSSI RAUMOLIN, Problems Related to the Transfer of Technology in the Mining Sector with Special Reference to Finland. 30.08.1988. 32 pp.

No 270 JUSSI KARKO, Factor Productivity and Technical Change in the Finnish Iron Foundry Industry, 1978-1985. 26.09.1988. 77 pp.

No 271 ERKKI KOSKELA, Timber Supply Incentives and Optimal Forest Taxation. 30.09.1988. 32 pp.

No 272 MIKAEL INGBERG, A Note on Cost of Capital Formulas. 07.10.1988. 29 pp.

No 273 JUSSI KARKO, Tuottavuuskehitys Suomen rautavalimoteollisuudessa 1978-1985. 10.10.1988. 38 s.

No 274 HILKKA TAIMIO, Taloudellinen kasvu ja kotitaloustuotanto - Katsaus kirjallisuuteen. 01.11.1988. 54 s.

No 275 MIKAEL INGBERG, Kapitalinkomstbeskattningens neutralitet i Finland. 11.11.1988. 32 s.

No 276 MIKAEL INGBERG, Näkökohtia metsäverotuksesta. 11.11.1988. 34 s.

No 277 MARKKU KOTILAINEN - TAPIO PEURA, Finland's Exchange Rate Regime and European Integration. 15.12.1988. 37 pp.

No 278 GEORGE F. RAY, The Finnish Economy in the Long Cycles. 20.12.1988. 104 pp.

No 279 PENTTI VARTIA - HENRI J. VARTIAINEN, Finnish Experiences in a Dual Trade Regime. 20.12.1988. 18 pp.

No 280 CHRISTIAN EDGREN, Tulorakenteen hyväksikäytöstä veronalaisen tulon kasvua arvioitaessa. 22.12.1988. 32 s.

No 281 PEKKA ILMAKUNNAS - HANNU TÖRMÄ, Structural Change of Factor Substitution in Finnish Manufacturing. 09.01.1989. 22 pp.

No 282 MARKKU RAHIALA - TIMO TERÄSVIRTA, Labour Hoarding Over the Business Cycle: Testing the Quadratic Adjustment Cost Hypothesis. 18.01.1989. 22 pp.

No 283 ILKKA SUSILUOTO, Helsingin seudun aluetalous panos-tuolostutkimuksen valossa. 08.02.1989. 27 s.

No 284 JAMEL BOUCELHAM - TIMO TERÄSVIRTA, How to Use Preliminary Values in Forecasting the Monthly Index of Industrial Production? 08.03.1989. 14 pp.

No 285 OLLE KRANTZ, Svensk ekonomisk förändring i ett långtidsperspektiv. 28.02.1989. 29 p.

No 286 TOR ERIKSSON - ANTTI SUVANTO - PENTTI VARTIA, Wage Setting in Finland. 20.03.1989. 77 p.

No 287 PEKKA ILMAKUNNAS, Tests of the Efficiency of Some Finnish Macroeconomic Forecasts: An Analysis of Forecast Revisions. 30.03.1989. 19 p.

No 288 PAAVO OKKO, Tuotantomuodon muutos ja sen merkitys yritys- ja aluerakenteelle. 08.05.1989. 14 s.

No 289 ESKO TORSTI, The Forecasting System in ETLA. 10.05.1989. 36 p.

No 290 ESKO TORSTI, MAT-ohjelmointitulkin käyttö ja rakenne. 11.05.1989. 67 s.

Elinkeinoelämän Tutkimuslaitoksen julkaisemat "Keskusteluaiheet" ovat raportteja alustavista tutkimustuloksista ja väliraportteja tekeillä olevista tutkimuksista. Tässä sarjassa julkaistuja monisteita on rajoitetusti saatavissa ETLAn kirjastosta tai ao. tutkijalta.

Papers in this series are reports on preliminary research results and on studies in progress; they can be obtained, on request, by the author's permission.

0033A/11.05.1989