SETL Newsletter Number 62 December 8, 1971

Kurt Maly

Final Specification Part of SETL and the Parser

In this newsletter we will supply the still missing modifications of the preparser, the lexical scanner, the various tables needed by them and finally the monitoring program which links all together.

The Preparser

Modifications*

In getkind we obviously have first to look for the lexical type of the token (e.g., 'return' as lexical type produces the kind 'var' whereas 'return' as token yields 'at', an operator).

```
        definef getkind(tokstat);preparse external symbkind,
        typkind;<type,token,->= tokdat; return if typkind(type)
        is x)ne Ω then x else if symbkind (token)is x ne Ω then
        x else 'var'; end getkind;
```

Previously the type of a node was a single item (e.g., '+') recorded in the function nodtype. Now we want to append to this type such data as <'ifend','scope control tokens'> or for a user defined operator <'uop','name of the operator'>. Hence, the type of a node is represented as a tuple and instead of the set function 'nodtype' we have now two user-defined functions 'nodtype' which has the same value as the original, and 'nodtypp' which returns the whole tuple.

```
        definef nodtype(node); preparser external typofnode;
        /* the set 'typofnode' replaces the old set 'nodtype' */
        (load) return hd typofnode(node); end;
        (store t) typofnode(node) = t;return;;
        end nodtype;
```

* Actual modifications of the code are indented.

```
definef nodtypp(node);preparser external typofnode;
return typofnode(node);end nodtypp;
```

The form of an item on statstak is: <state,kind,<lexicaltype, token,data if any>>, the third item of this tuple may be just a node name. At each 'iff' token we have to note the current number of items on statstak. This number will later be used by an algorithm, coded at the label 'users', to condense the 'iff' header.

```
newstate:  if token eq 'iff' then iffbeg=#statstak+2;;
           state=0b+...
```

According to new SETL conventions the part finders for an element of statstak should be:

```
definef kind stelt; return stelt(2); end kind;
definef tokof stelt; return stelt(4); end tokof;
definef tdat stelt; return stelt(3); end tdat;
```

To take care of the different nature of 'nodtype' we add another one:

```
definef tpdat stelt; return /* token plus token associated
data */stelt(3)(2:); end tpdat;
```

Since some operators can be used both as dyadic or monadic operators and have in those two instances different precedences we have to modify the precedence test (top of page 174).

```
condd = rprec(kind2) gt lprec(kind1);
condm = rprecl(kind2) gt lprecl(kind1);
/* rprecl and lprecl are the precedence tables with
the ambiguous operators thought of as monadic ones.  */
```

```
if(state and nnmask) eq zero then if condd then
go to getoken; else condensenn; end if condd; end if;

iff          (gross(kind  (2elm statstak))eq 'var')?
      (condd)?                        (condm)?
   getoken,(condensenon;),    getoken,(new=newat;
                                           condenseon;);

end iff;
newcycle:tokdat=new;
```

Again, because of 'nodtype' we have to rewrite the condensation procedures whereby we also will correct some existing errors.

In 'condensenn' the only modification is:

```
desc(new,2) = tokdat;nodtype(new) = <'catenation'>;
block condensenon;
tkdat=tpdat topoff statstak;oldel=tdat topoff
statstak; if nodtype(oldel) is z eq tkdat(1) and
#desc(oldel) gt 1 and z ne 'uop' then /* continuation
of same operation  */ desc(oldel,#desc(oldel)+1)=
tokdat;new=oldel;else desc(new,1)=oldel;
desc(new,2)=tokdat;nodtype(new)=tkdat;;
end condensenon;
block condenseon;
desc(new,1)=tokdat;nodtype(new)=tpdat
topoff statstak;
end condenseon;
block condense3(t);
desc(new,1)=tdat topoff statstak;oldel=tpdat
topoff statstak;nodtype(new)=
if oldel (2) eq Ω or t(2) eq Ω then <oldel(1)+t(1)>
```

```
else<oldel(1)+t(1), oldel(2)+t(2)>;
end condense3;
block condense4(t);
desc(new,2)=tdat topoff statstak;oldel=tpdat
topoff statstak; nodtype(new)=if oldel(2) eqΩ or
t(2) eqΩthen <oldel(1)+t(1)>else<oldel(1)+t(1),
oldel(2)+t(2)>;desc(new,1)=tdat topoff statstak;
end condense4;
```

To exit correctly with the root of the tree in case of 'er n er' we rewrite mispar:

```
mispar:dum =new;new=newat;if kind ne er
then desc(new,1)=tdat topoff statstak;nodtype(new)=<'missing
(,'+token>;else if gross(kind(2 elm statstak) eq er
then return dum;;<kind,tokdat>onto bakstak;
if gross(kind(3 elm statstak)) eq 'var' then condense4
(<',missing)'>);else condense3(<',missing)'>);end if gross;
end if kind; go to newcycle;
```

At parcas we make the following change:

```
parcas: new=newat;if gross(kind(3 elm statstak)
)eq 'var' then condense4(tokdat(2:));else
condense3(tokdat(2:));end if gross;go to newcycle;
```

Following the code for processing the iff-header. Before it is triggered by the single token <'semicolon','semicolon'> every 'node' (action-or test-) will already be condensed. Hence the binary tree will be represented by a sequence of alternating delimiters (',','?') and nodes (roots of subtree or just names) starting at a specified point in statstak('iffbeg'). To reconstruct that binary tree keep a fifo queue ('iffq') for the testnodes, take the next test node and affix the next two available nodes. Test nodes, encountered in this process, are added to the fifo queue. The process terminates when it catches up with the top of statstak. Whenever an error occurs the whole iff header is erased.

```
[users:]descl=nl;typofnodel=nl;<state,kind,
<'delimiter',';'>>onto statstak;iffend=#statstak;
cur=iffbeg;iffq=nl;/* we have to record the tree in
temporary sets since an error might occur */
descl(newat is root,1)=tdat statstak(cur);
nodtype(root)=<'∀∃'>;root stack iffq; cur=cur-2;
     /* beginning of main loop */
[contin:]cur=cur+4;if cur +3 gt iffend then go to
err;;(0≤∀n≤2) if tokof statstak(cur+1) eq ';' then
go to err;; if next iffq is top eq Ω then go to err;;
typel=tokof statstak(cur+1);type2=tokof statstak(cur+3);
go to if{<'∀∃','∀∃',ℓ1>,<'∀∃',',',ℓ2>,<',','∀∃',ℓ3>,
<',',',',ℓ4>,<',',';',ℓ4>}(typel,type2)is lab ne Ω
then lab else err;
      /*end of main loop*/
ℓ1: /* two test nodes */
descl(top,2)=newat is top;nodtypel(top)=<'catenation'>;
descl(top,1)=newat is topl;nodtypel(topl)=<'∀∃'>;
descl(topl,1)=tdat statstak(cur);topl stack iffq;
descl(top,2)=newat is top2;nodtypel(top2)=<'∀∃'>;
descl(top2,1)=tdat statstak(cur+2);top2 stack iffq;
go to contin;
ℓ2: /* one action node, one test node */
descl(top,2)=newat is top;nodtypel(top)=<','>;
descl(top,1)=newat is topl;nodtypel(topl)=<'∀∃'>;
descl(top,1)=tdat statstak(cur);topl stack iffq;
descl(top,2)=tdat statstak(cur+2);
go to contin;
```

```
ℓ3:  /* same as ℓ3 */
descl(top,2)=newat is top;nodtypel(top)=<',">;
descl(top,1)=tdat statstak(cur);
descl(top,2)=newat is topl;nodtypel(topl)=<'∀∃'>;
descl(topl,1)=tdat statstak(cur+2);topl stack iffq;
go to contin;
ℓ4:  /* two action nodes */
descl(top,2)=newat is top;nodtypel(top)=<',','>;
descl(top,1)=tdat statstak(cur);
descl(top,2)=tdat statstak(cur+2);
if type2 eq ';' and (iffend eq cur+3)then go
to headend; else go to contin;
[err:]print ' probable illegal or missing delimiter
in iff header'; (iffbeg≤∀n≤iffend)topoff
statstak;;tokdat=<'delimiter',';'>;kind=';';
state=hd top statstak;go to newstate;
[headend:](iffbeg≤∀n≤iffend)topoff statstak;;
new=root;<';',<'delimiter',';'>>onto bakstak;desc=
desc u  descl;typofnode=typofnode  u typofnodel;
go to newcycle;
definef nodetypel(node);preparser external typofnodel;
(load) return hd typofnodel(node);end;
(store t) typofnodel(node)=t;return;;
end nodtypel;
define  a stack  stk;stk(#stk+1)=a;return;end stack;
definef next queue;initial n=0;;n=n+1;return
queue(n);end next;
```

Because of operators having possibly two different precedences depending on their use, we have to create lprecl and rprecl. In order to do so we just make the corresponding portion of the code a macro (top of page 177):

```
[;lprec=analyse(lprinf);rprec=analyse(rprinf);
(∀x∈ hd[lprec] | rprec(x)  eq Ω)rprec(x)=lprec(x);;
```

```
(∀x∈kinds | lprec(x) eq Ω) lprec(x)=lprec(gross(x));
rprec(x)=rprec ((gross(x));;block precedence (lprec,
rprec,lprinf,rprinf);-]precedence(rprecl,lprecl,
lprinfl,rprinfl);
```

The setup routine has to be modified so that we can employ the 3-bit user portion for the 1-bit situation of 'semicolon'. The following replacements should be performed:

```
begend=<beg,nnon,beg,on,beg,f,parcas,beg,f,
 mispar,beg,specialcases,f,beg,f >;
bend=<f,nnon,f,on,f,f,parcas,f,f,mispar,
f,specialcases,f,users,f>;
starts=beg locsin begend;
finish={nnon,on,parcas,mispar,specialcases,
users}is labs locsin bend;
triple=<0,'semicol',0>
```

Errors

Corrections of errors can be found in D. B. Boyajian's master's thesis (page 107). Obvious misspellings and trivial errors are not mentioned.

Tables

First we give an informal table of the precedences. It should be noted that items listed under 'kind' are not all keywords. Some are tokens inserted by 'lex' (e.g., 'whl','foal',etc.). And some keywords are not listed at all because they get the classification 'var' (e.g., 'external','continue','quit',etc.).

kind	left precedence	right precedence
'∀∃', coma	1	2
'(', '[', '<', '{', lpar, if, atl, str, lod, initial, whl, foal, deff, defs, iff	3	27
var	4	4
';', semicol	5	5
label	6	6
while	7	7
then, when	8	8
else, doing	9	9
'='	10	10
at, store, define, definef, elop, go, eql, return	11	11
read, print, to	12	12
in	13	13
':'	15	15
','	16	16
'</', '<.', '>.', '>/'	17	17
ifex	18	18
thex	19	19
'∃', '∀'	21	21
'*', max, min, '/', '//', exp, log, with, less, lesf, u	24	24
'|', comp	25	14
is, as	25	22
elsex	25	20
'∋', hd, tl, '#', abs, bitr, floor, ceiling, dec, oct, hol, compile, type, atom, par, n, not, mop	25	25

eq, ne, lt, gt, le, ge, ∈, ne,	26	26
')', ']', '>', }, rpar, end	27	3
for dyadic use:		
'+', '-', uop	24	24
for monadic use:		
'+', '-', uop	25	25

Now follow the sets used by 'setup' to provide the various functions employed by the preparser.

```
tokinf={<tl[tilb(x)],tilb(x)(1),'0'>x∈{'∀∃',';','while',
        'then when','else doing','=','at store define definef
        go return','read print to','in',':',',','</ <.
        >. >/','∃ ∀', '∗  / // exp log max min
        with less lesf u ', '+ -','eq ne lt gt le ge ∈
        n∈ ','/','is as','∋ hd tl # abs bitr floor
        ceiling dec oct hol compile type
        n not}} u  {<x,'('>,x  separate('( [ < {
        if initial iff')} u {<x,')'>,x∈separate(
        ') ] > } end') };
typinf={<'comma','∀∃','o'>,<'name','var'>,<'skip','var'>,
        <'return','var'>} u {<x,x,'0'>,x∈separate(
        'label ifex thex elsex')} u {<x,'('>,x∈separate(
        'lpar atl str lod whl foal deff defs')}
       u {<'elop','at'>,<'eql','at'>,<'comp','|'>,
        <'mop','∋'>,<'rpar',')'>,<'semicol','semicol',
        'semicol'>,<'uop','+'>,<'ef','er'>};
lprinf={<tl x,hd x+2>,x∈tilb('( var ; label while
        then else = at read in')} u
                   {<tl x,hd x+14>,x∈tilb(': ,
        </ ifex thex')} u {<'∀∃',1>,<'elsex',25>,<'∃',21>,
        <'∗',24>,<'|',25>,<'+',24>,<'is',25>,
        <'∋',25>,<'eq',26>,<')',27>};
```

```
rprinf={<'er',1>,<'∀∃',2>,<')',3>,<'|',14>,<'is',22>,
       <'(',27>};
lprinfl=lprinf lesf '+' with <'+',25>;
rprinfl=nl;
definef separate(x);y=x;res=nl;
(while 1≤∃[k]≤#x | x(k) eq ' ' or k eq#y)
<res,y>=<res with y(1:if k lt#y then k-1 else k),y(k+1:)>;
end while; return res; end separate;
```

The Lexical Scanner

Modifications

```
       definef scan;preparser external separate;
       initial endset={<'namop','name'>,<'intrealbit',
       'integer'>,<'bitoct','bitstring'),<'octbit','bitstring'>,
       <'intreal','integer'>,<'real','real'>,<'quoted',
       'string'>,<'next','delimiter'> ;er='$';
       s=<input,1>;cstring=record(s);if cstring eq
       nulc or cstring eqΩ then return<'ef','ef'>;;
       cstring(#cstring+1)=er;n=1;print cstring;
       nullop=separate('newat null nl nult nulc
       nulb t f true false holl');
       operator=separate('hd tl abs bitr floor ceiling
       not n dec oct hol compile type atom pair
       as in eq ne lt gt le ge max min exp log
       with less lesf and or nε u is');....

       switch: go to {<'end',endc>,<'endo', endoc>,<'endb',
       endbc>,<'endl',endlc>,<'go',goc>,<'skip',loop>,
       <'cont',contc>,<'do',doc>} (hd action);
       endoc:[;dum =bit(token(datum(2)is x+2:#token-x-1));
       /* bit is an internal function which converts a
       characterstring representing an octal number to a
       characterstring representing this same number as a
       bitstring */ datum=(datum(1)+dum) as bstring;
```

```
ltype='bitstring';block bitoctal;-] go to endlc;
endbc:[;datum=(datum(1)+token(datum(2)is x+2;
#token-x-1)) as bstring;ltype='bitstring';block
octalbit;-] go to endlc;
endc:ltype=endset(state);
endlc:n=nn;...
```

The Tables

Actual reserved words are only the following ones:

if, initial, iff, then, when, while, else, doing at, store, define, definef, load, block, go, to, return, read, print, til, end, external, local, continue, quit, pow, npow, record, and the following delimiters:

([< { ; = : , </ <. >. >/ ∃ ∀ * / // | ∋ #

)] > + - $ ∀∃ and the . in real numbers.

The names of the members of 'nullop', 'objtype' and 'operator' are not keywords_- they can be used as variable names but may not be used as user-defined operators (i.e., underlined). *

The lexical types are:	e.g.
name	setl
operator	max
delimiter	{
bitstring	1b76o (internally 111110)
string	'character string'
integer	25
real	2.5
nullop	newat
objtype	integer
uop	useroperator
er	$
ef	$$

* Whenever a user-defined operator is used as name it should not be underlined.(e.g., define a op b;...end op; y=a op c; x=op;). There are at the moment no user-defined operators with no arguments allowed.

Instead of giving the action table and the supplementary routines in its formal form expected by the setup routine, we will present the routines as a piece of code and the table as a matrix.

Code for the various actions:

```
ob1:        bitn=t;
ob2:        bitn=f;
er2:        print 'illegal period';
relbra:     if cstring(nn+1) is c eq '/' and cstring(nn+2) ne ' '
            or c eq '.' then token=cstring(nn)+c;nn=nn+2;
            /* token is relational symbol */ else token=cstring(nn);
            nn=nn+1;;
qm:         if cstring(nn+1) is c eq '⌉' then  /* token is question
            mark*/ token=cstring(nn)+c;nn=nn+2; else
            token=cstring(nn);nn=nn+1;;
barcom:     if cstring(nn+1) is c eq '/' then token=cstring(nn)+c;
            nn=nn+2; else if c eq '*' then nn=endcomment( );
            action='skip';else token=cstring(nn);nn=nn+1;
            end if c;end if;
skip1:      nn=nn+1;
eof:        token=cstring(nn);ltype='ef';nn=nn+1;
specend:    token=cstring(nn);nn=nn+1;
qtest:      nn=nn+1;if cstring(nn) ne '''' then action='end';;
ercheck:    cstring=record(s);if cstring eq nulc or cstring eq Ω
            then if token eq nulc then token='ef';ltype='ef';
            action='endl';else action='end';cstring(1)=er;
            nn=1;end if token;else print cstring;cstring(#cstring+1)=
            er;nn=0;end if cstring;
octend:     bitoctal;token=token+cstring(nn);
bitend:     octalbit;token=token+cstring(nn);
```

```
er3:        print 'illegal bitstring specification';
octo:       bitn=f;
bita:       if n bitn then print 'illegal bitstring specification';
            action='end'; else datum=<token,#token>;;
oct:        datum=<bit(token),#token>;
opcheck:    token=token+'.';
            iff           (token∈nullop)?
               (ltype='nullop';),      (token ∈operator)?
                      (ltype='operator';),(token ∈{'input','output'}) ?
                      (ltype='name';),    abret?
                                               act1,act2;
            abret:={<'om','Ω'>,<'nm','#'>,<'el','∈'>,<'fa','∀'>
            <'ex','∃'>,<'st','|'>,<'an','∋'>,<'qm','∀∃'>}
            (token)is tok ne Ω; /* it should be noted that this
            option, i.e., allowing to write fa for the symbol
            '∀', etc., restricts the user to not defining an
            operator of this name */
            act1: ltype='delimiter';token=tok;
            act2: ltype='uop',datum=token(1:#token-1);
                  token='uop';end iff;

definef endcomment;scan external cstring,nn;nn=nn+2;
advance:(while n(cstring(nn)is c)∈{'*',er)nn=nn+1;;
go to {<'*',star>,<er,eror>} (c);[star:]nn=nn+1;if cstring
(nn) eq '/' then return nn; else go to advance;;
            [eror:]cstring=record(s);
if cstring eq nulc or cstring eq Ω then print 'illegally
structured comment';cstring(1)=er;nn=0;return nn;else nn=1;
print cstring;cstring(#cstring+1)=er;go to advance;
end endcomment;
```

symbols state	a a...2 except o,b	o o	b b	1 0,1	2 2,3,4, 5,6,7	8 8,9	. .	> >,<	∀ ∀	/ /	' '	er $	bl blank	* +,-,*,(,),=,∩, [,],:,#,∈,∃, |,},{,∋,;,','
namop	cont	cont	cont	cont	cont	cont	do op-check endl		end	end	end	do er-check cont	end	end
intreal-bit	end	do oct go oct-bit	do bita go bit-oct	cont	do octo cont	go int-real	go real	end	end	end	end	do er-check cont	end	end
bitoct	endo	do oct-end endl	endo	cont	cont	do er3 end	endo	endo	endo	endo	endo	do er-check cont	endo	endo
octbit	endb	endb	do bit-end endl	cont	do er3 end	do er3 end	endb	endb	endb	endb	endb	do er-check cont	endb	endb
intreal	end	end	end	cont	cont	cont	go real	end	end	end	end	do er-check cont	end	end
real	end	end	end	cont	cont	cont	end	end	end	end	end	do er-check cont	end	end
quoted	cont	cont	cont	cont	cont	cont	cont	cont	cont	cont	do qtest cont	do er-check cont	end	end
next	go namop	go namop	go nam-op	do ob1 go int-real-bit	do ob2 go int-real-bit	go int-real	do er2	do rel-bra end	do qm end	do bar-com end	do skipl go quoted	do eof endl	skip	do specend end

Keypunch conventions

SETL character set	CDC 64-character set		026 keypunch
	029 keypunch	optional	multiple punch
A	A		
B	B		
C	C		
.			
.			
.			
.			
Z	Z		
0	0		
1	1		
.	.		
.	.		
.	.		
9	9		
±	+		
-	-		
*	*		
/	/		
(	(		
)	)		
$	$		
=	=		
blank	blank		
,	,		
.	.		
Ω	≡	OM.	0-8-6
[	[		8-7
]	]		0-8-2
:	:		8-2
#	↓	NM.	11-8-6
ε	→	EL.	0-8-5

∀	∨	FA.	11-0	
∃	∧	EX.	0-8-7	
|	↑	ST.	11-8-5	
'	#		8-4	
<	<		12-0	*)
>	>		11-8-7	*)
{	≤		8-5	
}	≥		12-8-5	
∋	¬	AN.	12-8-6	
;	;		12-8-7	
EOL	EOL		EOL	

double character symbols:

>	>.		**)
<	<.		**)
≥	>/		
≤	</		
?('∀∃')	∨∧	QM.	

*) <, > as used for tuples.

**) <, > as search delimiters in iteration header.

The Parser

```
definef parser; lex( );treetop=preparser( );return postparse
('program',treetop,0);end parser;
definef nextoken;lex external add; add external lexl;
initial n=0;;n=n+l;return lexl(n);end nextoken;
```

Errors in Newsletters 58, 58a, 61.

Newsletter 58.

Page	Line	
3	0	20. return on page 14 under label 'return'. Remarks:return as in a subroutine is marked as such.

4	7	initial elop=<'elop','elop'>;lcl=<'local','local'>;
		extnl=<'external','external'>;mns=<'mop','mop'>;
		mnskip=<'skip','-'>;lpar=<'lpar','lpar'>;
		rpar=<'rpar','rpar'>;foal=<'foal','foal'>;
		whl=<'whl','whl'>; atl=<'atl','atl'>;
		lod=<'lod','lod'>;str=<'str','str'>;
		comp=<'comp','comp'>;deff=<'deff','deff'>;
		defs=<'defs','defs'>;endv=<'name','end'>;
		semi=<'delimiter',';'>;label=<'label','label'>;
		com=<'coma','coma'>;semicol=<'semicol',
		'semicol'>;eql=<'eql','eql'>;ret=<'return',
		'return'>;exp=<'expr','expr'>;...

9	27	.... <'quit',conquit>,<'return',retrn> ;
9	30	rel={'<.','</','>.','>/'};
10	23	if ct le 5 then contok=contok+if token(1) ne 'uop' then <tok> else <token(3)>;;
11	18	... 'end' then token= nt;/* check....
13	12	add(token);if token(1) eq 'lpar' then beg=t;; go to getok;

14	14	retr:new=nt;if new(2) eq ';' then add
		(ret);else add(token);;new stack tokstack;
		go to getok;

15	11	quit;;if a(i) eq 'uop' or b(i) eq 'uop' then continue;;

Newsletter 58a.

Page	Line	
2	15	...partc=1; ifswi=f;
2	19	(if token(2) eq '∀' then if n itfd then if n ifswi then pnum=part-1;partc=1; itfd=t;end if n ifswi;end if n itfd;end if token;),...
2	27	...add(foal+<contok>);
3	12	(pnum<∀n≤#itsta)...
3	12	...add(foal+<contok>);
3	28	...add(whl+<contok>);

Newsletter 61a.

Page	Line	
1	22	...end. While writing this grammar it was discovered that the macro ':number' for 'partof(node, locater)' is ambiguous (e.g., desc(:1,2)(1:2)) Since from the context it is (hopefully) clear when ':number' is this macro it has not been changed here. But hereafter it should be written as '** number'.

3	17	...else ok2=t;;proceds(#proceds)(6)=t;
		/t2/ if n ok2 then ok=f;;proceds(proceds)(6)=f;
		;...

3	29	...,newat is defproc,f>...
4	9	...,newat is defproc,f>...
9	23	statement */ [=:'='('<>'(','[name].2)'OMITTED'/ multiple name initialization. /8,8,2,2

```
                 /t2/ok=(topof proceds)(6);,'illegal position
                 of initialization statement'
                 =: '=' ('<>'...

4        33         /a2/statseq;
                 =: '='[name]'OMITTED'/name initialization
                    statement./ 8,2
                    /t2/ ok=(topof proceds)(6);,'illegal
                    position of initialization statement'
                 =: 'IN'exp...

5        0       =: 'READ' cname. /  read statement./ 10
                     /a2/ statseq;
5        3           /a2/ statseq;
                 =: 'PRINT' cexp./ print statement./ 10
                     /a2/ statseq;
                 =: ]'PRINT'...

5        10      ...symbol(desc(:1,1));
                 =: 'CATENATION'[name]'()'/ call tosubroutine
                    with no arguments./8,2
                    /a2/statseq;symbol(desc(:1,1));
                 =: 'UOP'.[uop]expl.2/dyadic operator call./
                    /a2/statseq;symbol(nodtypp(:1) );
                 =: 'UOP'.[uop]expl. /monadic operator call./
                    /a2/statseq;symbol(nodtypp (:1));

6        20      =: 'RETURN'.[return]

7        18      ...nodtypp(:2)(3) is pname,/  typofnode
                 (node) is a pair, nodtype returns the head
                 nodtypp returns the whole tuple /
                 if nodtype...
```

```
8      7     =:{'>/','>.','<.','</'}exp.({'>/',
               '>.','<.','</'}

8      9     ... ∈{'</','<.'}...

8      10          ∈{'</','<.'}....  ∈ {'>/','>.'};

10     12    ...     'NE'} {'SET',...

10     14                      ... /8,2
             =: {'EQ','NE'}   'Ω' exp./  omega test./ 8,8
             =:         exp. {'SET','INTEGER',...

10     17
                            .../8,2
             =: {'EQ','NE'} exp. 'Ω'/omega test./8,8

10     23           ... (nodtypp(:1));

10     26           ... (nodtypp(:1));..

10     33    =:  'AS' exp.[objtype]

10     35    eliminate line

11     20    =: [bitstring]
                /a2/data(dt:1);/* store octal
                constant */ definef dtx;return x(3);end dt;

11     25    ...    [nullop]
             =: 'CATENATION'[name] '()'/call to
                function with no arguments. /8,2
                /a2/symbol(desc(:1,1));

13     9     =: '()' ({'POW','RECORD'} exp.)/ built in
                function./8,8
```