SETL Newsletter Number 19

February 26, 1971
Kurt Maly

Lexical Description of SETL:

1. Lexical Scanner

```
definef nextoken;
      initially setup(type, table, rpak, cstring); n=1;;
state = nxt; nn = n-1; data =/\; token = nulc;
loop: nn = n+1; action = table(state, type(cstring(nn))):
switch: go to {<end,endc>,<endl, endcl>, <go, goc>,<skip, loop>,
              <cont, contc>,<do, doc>} (hd action);
goc:  state = tl action;
conc:  token = token+cstring(nn); go to loop:
endc:  ltype = enst(state);
endcl:  n = nn; return if data ne /\ then <ltype, token, data> else
                                          <ltype, token>;
doc:  <-, rout, action> = action; rpak(rout);
      go to if action eq /\ then loop else switch;
end  nextoken;
```

This program will extract from the input string (cstring) the next token and give it one of the following lexical types classification.

Lexical types:	name	(e.g.	setl)
	operator	(e.g.	op)
	delimiter	(e.g.	{)
	charstri	(e.g.	"character string")
	bitstri	(e.g.	01b056o)
	integer	(e.g.	129)
	octal	(e.g.	760)

er	(e.g.	er)
ef	(e.g.	er er)

Together with the actual token (and where applicable additional data) the lexical type of the current token will be returned to the calling program.

2. Character Types

The representation of the fourteen different character types acceptable for setup is:

[a, o, b, l, 7, 8, +, gt, co, ., (,), bl, er]

their exact description is given by the following set:

{⟨a, 'acdefghijklmnpqrstuvwxyz'⟩,
⟨o, 'o'⟩,
⟨b, 'b'⟩,
⟨1, '01"⟩,
⟨7, '234567'⟩,
⟨8, '89'⟩,
⟨+, '-+/*S'⟩,
⟨gt, ''''⟩,
⟨co, ','⟩,
⟨., ','⟩,
⟨(,'('⟩,
⟨),')'⟩,
⟨bl, ' '⟩,
⟨er; '='⟩}

This is a set of ordered pairs where the first item of the ordered pair is a character type whereas the second item is a string of characters which belong to that character type.

3. State Transition Table

{⟨namop , [cont,cont,cont,cont,end,end,end,⟨do,opcheck, endl⟩,
end,end,cont,cont,end, end]⟩,
⟨charstri, [cont,cont,cont,cont,⟨do,ocheck⟩,cont,cont,cont,
cont,cont,cont,cont,cont, cont]⟩,
⟨numbit, [⟨do,er(1),endl⟩,cont,⟨go,numbit⟩,⟨go,int⟩, end,end,end,end,
end,end,⟨do,endo,endl⟩,⟨go,bitoct⟩,end, end] ,
⟨bitoct, [⟨do,er(1),endl⟩,cont,cont,⟨do,er(1),endl⟩,end,end,end,end,
end,end,end,⟨do,er(1),endl⟩, end, end] ,
⟨numbit, [⟨do,er(1),endl⟩,cont,cont,⟨go,int⟩,end,end,end,end,
end,end,⟨do,endo,endl⟩, end, end] ,
⟨int, [⟨do,er(1), endl⟩, cont,cont,cont,end,end,end,end,
end,end,⟨do,er(1),endl⟩,⟨do,er(1), endl⟩,end, end] ,
⟨nxt, [⟨go,namop ,⟨go,numbit⟩,⟨go,numbit⟩,⟨go,numbit⟩,
⟨do,spend,endl⟩,⟨do,skip,go,charstri⟩,⟨do,com,endl⟩,
⟨do,per,endl⟩,⟨do,bra,endl⟩,⟨oco,bral,endl⟩,
⟨go,namop⟩,⟨go,namop⟩,skip,⟨do,endr,endl⟩]⟩}

This table is a set of ordered pairs, the first item of each designates a lexical state and the second a list of actions to be taken for each character type.

4. Auxiliary Routines

```
{⟨opcheck, ' token=token+"."; nn=nn+1; if ops(token) ne ⌐
            then token = ops(token); ctype="delimiter";
            else ctype = "operator";; '⟩,
⟨endo, ' ltype = "octal"; '⟩,
⟨spend, ' token = cstring(nn); nn=nn+1; ltype = "delimiter"; '⟩,
⟨com, ' ltype = "delimiter"; if cstring(nn+1) eq "."
            nn=nn+2; else token = ','; nn=nn+1;; '⟩,
⟨per, ' ltype = "delimiter"; if cstring(nn+1) eq "."
            nn=nn+2, else token = "."; nn=nn+1;; '⟩,
⟨bra, ' ltype = "delimiter"; x=cstring(nn+1) + cstring(nn+2);
            if braset(x) eq ⌐ then token=cstring(nn); nn=nn+1;
            else token = braset(x); nn=nn+1;;'⟩,
⟨bral, ' ltype = "delimiter"; x = cstring(nn+1)+cstring(nn+2);
            if bralset(x) eq ⌐ then token=cstring(nn); nn=nn+1;
            else token = bralset(x); nn=nn+3;;'⟩,
⟨gcheck, ' nn=nn+1; action = if cstring(nn) eq ''''' then
            "cont" else "end"; '⟩,
⟨endr, ' if cstring(nn+1) ne "=" then nn=nn+1; ltype="delimiter"

            else if(cstring(nn+2) eq "=") and(cstring(nn+3)eq "=")
            then ltype="ef"; token="er er"; nn=nn+ ;
            else ltype="er"; token="er"; nn=nn+2;; end if; '⟩,
⟨er'1', 'print("illegal number"); print(token);'⟩}
```

5. Diverse Functions

The function endst is used in nextoken; its definition is:

```
endst = {⟨'namop', 'name ' ,⟨'numbit','integer'⟩,
```

⟨'bitoct', 'bitstri'⟩, '

⟨'int', 'integer'⟩}

The functions ops, braset and bralset are additional sets used in rpak.

ops = {⟨'om.', '≡'⟩,⟨' mm.','↓'⟩,⟨'e.','→'⟩,
⟨'el.','¬'⟩,⟨'fa.','∨'⟩,⟨'ex.','∧'⟩,⟨'st.','↑'⟩}

braset = {⟨'+(','<'⟩,⟨'*(','≤'⟩,⟨'.(','['⟩}

bralset = {⟨'+)','>'⟩,⟨'*)','≥'⟩,⟨'.)',']'⟩}