T -> T . mul_op F       OW reduce (by 3, recognizing E)
T -> T mul_op . F       on ( shift and goto 8
F -> . ( E )            on ID shift and reduce (by 1, recognizing F)
F -> . LITERAL          on LITERAL shift and reduce (by 1, recognizing F)
---------
SL -> SL . S            on S shift and reduce (by 2, recognizing SL)

0 stmt_list 1 WRITE 4                         sum / 2 ...
0 stmt_list 1 WRITE 4                         T write sum / ...
0 stmt_list 1                                 S write sum write ...
0 stmt_list 1                                 S sum := ...

mult_op       ->  * | /

---

Find_Insert (stack XN ... X2 X1, token a) returns (string ins)
Cost C(t) for each terminal.  Higher C(t) means t is less likely to
insert-correctable language.  It also requires buffering of epsilon
Modify the input stream and re-start parsing.  Whenever we want to
it's in FOLLOW(FT) but not in CS-FOLLOW(FT).
S -> id := E.  We match the id, leaving us with
- The parse table predicts FT ->
- We have a mismatch between := in the stack and * on the input.
  - The parse table predicts FT ->
- We have a mismatch between ) in the stack and X on the input.  We
  - The parse table predicts FT ->

Much easier, when we predict an epsilon production, to remember that we
parsing:
-- FOLLOW sets for all symbols:
for all non-terminals X
EPS(c) := false;  FIRST(c) := {c}
-- EPS values and FIRST sets for all symbols:
Put another way, ')' is in FOLLOW(E) in the context where E was
But the fact that ')' is in FOLLOW(E) does
Now consider the production S -> write E.
the grammar.
---
PREDICT(A ->
----------
when you're looking at a single production to decide on the top of the
You can see this in the case of a production A -> B C D:
(a) =={c: c ∈ FIRST(B) and c != $$}
(b) ==$$
(c) ==FIRST(B) U FOLLOW(B) U {$$}
(d) == EPS(FOLLOW(B))
(e) == EPS($$)
(f) == EPS(B)
(g) == EPS(c)
(h) == $$
(i) ==$$

*** In a recursive descent parser, if c
lower case letters near the end of the alphabet for _strings_ of terminals
=>* means "derives in zero or more steps"
that can follow the LHS anywhere in the grammar predicts the epsilon
- match any that are terminals
- The scanner is a subroutine (function) called by the parser.
- Sometimes need semantic information in order to scan (yuck).
- The power of bottom-up parsing comes from its ability to recognize
- basically a finite automaton with a stack
- makes all decisions based on input and top-of-stack symbol
- order to parse top-down left-to-right, we end up with a tree that tends

Also nice if the parse trees reflect semantic structure, but that's not
---
---
---
---
---
---
---