**T** → **T** . mul_op **F**       OW reduce (by 3, recognizing **E**)
**F** → . LITERAL          on LITERAL shift and reduce (by 1, recognizing **F**)
**F** → . ( **E** )            on ( shift and goto 8
**E** → . **E** add_op **T**
**E** → . **T**                on **T** shift and goto 7
**F** → . ID               on ID shift and reduce (by 1, recognizing **F**)
**T** → . **T** mul_op **F**

1:
0 stmt_list 1 WRITE 4 expr 6                  $$
0 stmt_list 1 WRITE 4 term 7 / 11             2 $$
0 stmt_list 1                                 S write sum / ...
0 stmt_list 1 ID 2 := 5 expr 9 + 10 term 13   write sum write ...
0 stmt_list 1 ID 2 := 5 expr 9 + 10           T write sum ...
0 stmt_list 1 ID 2 := 5 expr 9 + 10
0 stmt_list 1 WRITE [ID 2 := 5] expr 9 + 10
0 stmt_list 1 WRITE [ID 2 := 5] expr 9 + 10
0 stmt_list 1 WRITE [ID 2 := 5]expr 9 + 10
0 stmt_list 1 WRITE [ID 2 := 5]expr 9 + 10
0 stmt_list 1 WRITE [ID 2 := 5]expr 9 + 10
0 stmt_list 1 WRITE [ID 2 := 5]expr 9 + 10
0 stmt_list 1 WRITE [ID 2 := 5]expr 9 + 10

PDAs).  A PDA can be specified with a state diagram and a stack.  The
A scanner is a DFA.  It can be specified with a state diagram, as
Like a table-driven LL parser, an LR parser uses a big loop in which it
Allowing deletions, too, makes for better quality repairs, and works for
\[ \text{ins} := \text{??} \]
\[ \text{E table is moderately large, but manageable} \]
\[ \text{else E(A,a) = lowest cost prefix w of a in A; that is,} \]
\[ \text{if A does not derive a string containing a, then} \]
Cost C(t) for each terminal.  Higher C(t) means t is less likely to
- linear time and space requirements
- most programming languages have LL grammars
- never go back to a previous input position
- in special cases.

Fig. 2.23 shows the generated FIRST, FOLLOW, and PREDICT sets
with the same lhs, then the grammar is not LL(1).
How to deduce?
ANTLR, by default, uses global FOLLOW sets and Java/C++/C# exception
get_next_token()
and soon run into trouble again when we see * instead of := on the
T, and then F.  F will predict F → id, match X, then return

Several error-recovery mechanisms, including the version of Wirth’s
To delete to what we hope is the end -- that is, to write the simpler
else get_next_token()
stmt()          -- try again
loop
- Subroutines for strings, similar to the inner loop above:
Fig. 2.23 shows the generated FIRST, FOLLOW, and PREDICT sets
with the same lhs, then the grammar is not LL(1).
How to deduce?
ANTLR, by default, uses global FOLLOW sets and Java/C++/C# exception
get_next_token()