Semantic Error Recovery

Check ST & ST-2 (oops)
semError (int reqd)
NO codegen -- instead
ST := ST - 2 and insert
the following
expr. erroreentry
to get

We complain if we see a new error.
Semantic Error Recovery

\[ a := z \times (x + y) . \]

Check ST & ST-2
(Previous error)
\[ \text{semError (int reqd)} \]

NO codegen -- instead
ST := ST - 2 and insert
the following

\[ \text{expr. errorentry} \]
\[ \text{mulop } * \]
\[ \text{expr } z \ldots \text{int} \]
\[ \text{expr } a \ldots \text{int} \]

We don’t complain if we see an old error, but we propagate it if possible.
Semantic Error Recovery

\[ a := z \times (x + y) \cdot ; \]

Check ST & ST-1
(previous error)
\[ \text{semError (int reqd)} \]
NO codegen -- instead
\[ ST := ST - 2 \]

We don’t leave a special entry because the genStore protocol doesn’t leave any entries at all.

expr. erroreentry

expr a ... int
Semantic Error Recovery

Rules:
(a) Check for previous errors (special entry) if seen, tear down & propagate - silently
(b) Check for current errors (illegal types...) if seen, tear down & propagate - complaining
(c) Otherwise, do the right thing.

Note: When tearing down, propagate any error to some remaining entry. If none remain the evidence of the error disappears from the run.
Notes:
(a) Only one error message is issued.
(b) All other action routines have a normal input protocol - except for the special entry.
(c) Eventually the error entry disappears and you are back to normal processing.
Semantic Error Recovery

Hint:

1. Build procedures to do this checking. Otherwise your logic will get convoluted.
2. Pass entry numbers to these procedures.
3. You can also build two layers (or more) of procedures: genAdd calls “okForArithmetic” which calls procedures at a lower layer. “okForArithmetic” can be reused in genMult, and the lower layer procedures can be used by several of the first layer procedures.