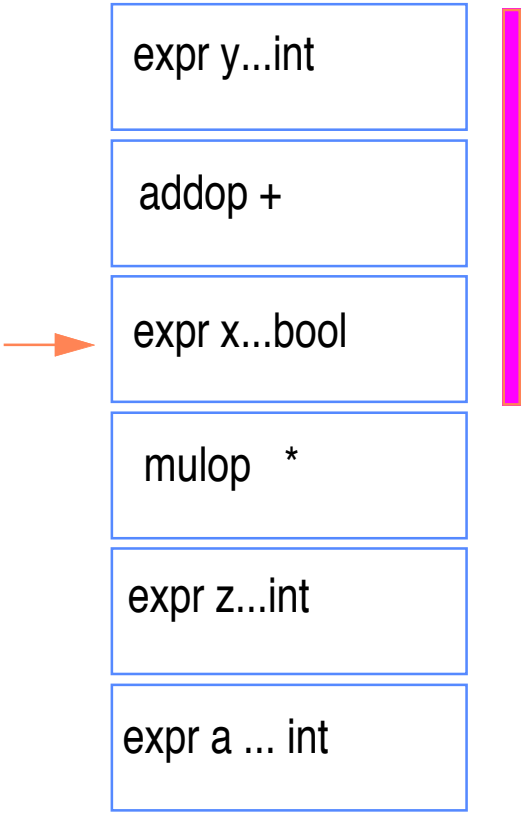
Semantic Error Recovery

a := z * (x + y) • ;

Represents the input params to the current routine



Check 2 exprs (oops--int required here)

semError (int reqd)

NO codegen -- instead return the following

expr. errorentry

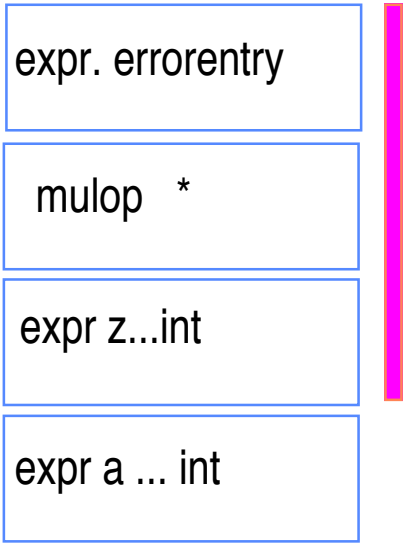
to get

We complain if we see a new error.

AddExpr

Semantic Error Recovery

a := z * (x + y) • ;



Check 2 exprs
(previous error)

~~semError (int reqd)~~

NO codegen -- instead
return the following

to get →

We don't complain if we see an old error, but we propagate it if possible.

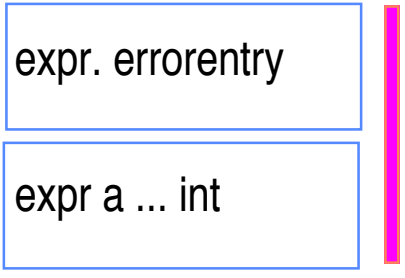
MultExpr

Semantic Error Recovery

a := z * (x + y) • ;

Check ST & ST-1
(previous error)
~~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.



Semantic Error Recovery

Rules:

(a) Check for previous errors (special entry)

if seen, - silently return error

(b) Check for current errors (illegal types...)

if seen, - complain and return error

(c) Otherwise, do the right thing.

Note: We only return an expression error if the function normally returns an expr. If not, the evidence of the error disappears from the run.

Semantic Error Recovery

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. You can also build two layers (or more) of procedures: The high level procs call the low level ones. You can build an equal method into the type descriptors to help with this.