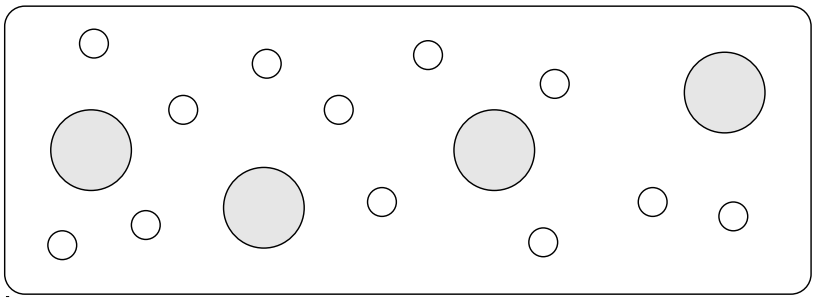
Error repair



Good programming languages are designed with a relatively large "distance" between syntactically correct programs, to increase the likelihood that conceptual mistakes are caught as syntactic errors.

Error repair usually occurs at two levels:

Local: repairs mistakes with little global import, such as missing semicolons and undeclared variables.

Scope: repairs the program text so that scopes are correct. Errors of this kind include unbalanced parentheses and begin/end blocks.

Repair actions can be divided into *insertions* and *deletions*. Typically the compiler will use some lookahead and backtracking in attempting to make progress in the parse. There is great variation among compilers, though some languages (PL/C) carry a tradition of good error repair. Goals of error repair include:

1. No input should cause the compiler to collapse.
2. Illegal constructs are flagged.
3. Frequently occurring errors are repaired gracefully.
4. Minimal stuttering or cascading of errors.

LL-style parsing lends itself well to error repair, since the compiler uses the grammar's rules to *predict* what should occur next in the input.

Augmenting recursive descent parsers for error recovery

Recursive and LL parsers are often called *predictive*, because they operate by predicting the next step in a derivation.

Suppose the parser is operating in procedure `A` for some nonterminal A. If an error occurs, it seems reasonable to recover by skipping to a symbol that could follow A, and then return.

E	$\rightarrow$	**T** E'
E'	$\rightarrow$	**+ T** E'
E'	$\rightarrow$	**- T** E'
	$\mid$	λ
T	$\rightarrow$	**F** T'
T'	$\rightarrow$	$*$ **F** T'
T'	$\rightarrow$	**/ F** T'
	$\mid$	λ
F	$\rightarrow$	**(E)**
	$\mid$	**a**

	First	Follow
E	$\{(, a\}$	$\{), \$\}$
E'	$\{+, -\}$	$\{), \$\}$
T	$\{(, a\}$	$\{+, -,), \$\}$
T'	$\{*, /\}$	$\{+, -,), \$\}$
F	$\{(, a\}$	$\{*, /, +, -,), \$\}$

Procedure $E'(StopSet)$
  if $(LookAhead(+))$ **then**
    call $Expect(+)$
    call $T(\{+, -\} \cup StopSet)$
    call $E'(StopSet)$
  else
    if $(LookAhead(\$, \text{')'}))$ **then**
      return ()
    else
      call $ErrorRecover(StopSet)$
    fi
  fi
end