Passing

* LL : Left-to-right, Leftmost derivation
  (input read)
  - simpler, easier to understand & generate
  - top-down, predictive parsers

* LR : Left-to-right, Rightmost derivation
  (input read)
  - larger, more intuitive
  - bottom-up parsers
  - construct the parse tree from leaves up
  - no prediction (shift-reduce parsers)

Example: pp. 62-63, Figure 2.13

* LL(1), LR(2)

\[ \uparrow \quad \uparrow \]
how many tokens of lookahead are
required in order to parse.
\Rightarrow most compilers use only one token
look ahead.

Writing an LL(1) grammar:

Two obstacles: 1) Left recursion
Not having these
2) common prefixes
\Rightarrow does not guarantee
LL(1), but required.
\* A → A + B \quad \text{left recursion}
\* A → C + B \quad \text{indirect left recursion}
\* C → A

\* A → aB \quad \text{common prefix}
\* A → a

\* \underline{Left Factoring: (Getting rid of Common prefixes)}

A → aB
A → a

A → aB \mid a

A → a (B \mid E)

A → a A_{-\text{new}}
A_{-\text{new}} → B \mid E

\* \underline{Eliminating Left Recursion:}

A → A + B
A → a
B → b

A → a + B
B → b + B
B → b

\* Example: pp. 64-65, Figure 2-14