RECURSIVE DESCENT PARSING

Recursive-descent parsing is a top-down method of syntax analysis in which a set of recursive procedures is used to process the input. One procedure is associated with each nonterminal of a grammar. Execution begins with the procedure for the start symbol, which halts and announces success if its procedure body scans the entire input string.

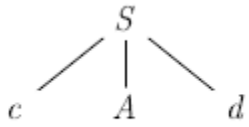
General recursive-descent may require backtracking; that is, it may require repeated scans over the input. However, backtracking is rarely needed to parse programming language constructs, so backtracking parsers are not seen frequently.

For Example: Consider the grammar

$$S \rightarrow c\ A\ d$$

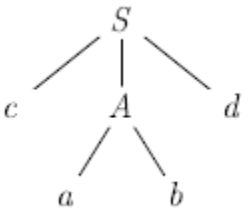
$$A \rightarrow a\ b \mid a$$

To construct a parse tree top-down for the input string w = cad, begin with a tree consisting of a single node labeled S, and the input pointer pointing to c, the first symbol of w. S has only one production, so we use it to expand S and obtain the following tree.



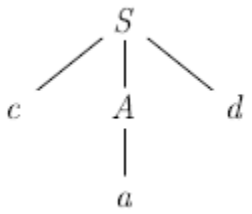
The leftmost leaf, labeled c, matches the first symbol of input w, so we advance the input pointer to a, the second symbol of w, and consider the next leaf, labeled A.

Now, we expand A using the first alternative A → ab to obtain the following tree.



We have a match for the second input symbol, a, so we advance the input pointer to d, the third input symbol, and compare d against the next leaf, labeled b. Since b does not match d, we report failure and go back to A to see whether there is another alternative for A that has not been tried, but that might produce a match.

In going back to A, we must reset the input pointer to position 2, the position it had when we first came to A, which means that the procedure for A must store the input pointer in a local variable. The second alternative for A produces the following tree.



The leaf a matches the second symbol of w and the leaf d matches the third symbol. Since we have produced a parse tree for w, we halt and announce successful completion of parsing.