Top Down Recursive Descent Parser

Write a Top Down Recursive Descent Parser for CL, which uses your lexical analyzer to obtain tokens. The parser should generate a parse tree for the input, and should report errors in a user-friendly fashion.

Parse tree nodes can conveniently be built, using a routine Node(). The arguments to Node are: the grammar RULE NUMBER (for this purpose, you should give distinct numbers to each alternative rule in the grammar), followed by the pointers to the child nodes of this rule, in left to right order. Unused child nodes should be given pointer values of 0. Hence, node numbers must be >0, and I’d suggest using small integers (array indices). Node returns the number of the node it builds. Some of the node fields usually used for “child” pointers can instead be used for Tok.Id values, since the rule number in the node allows unambiguous interpretation of each other field in that node.

For the current problem, print each node, one per line, in the form: <node number>: <rule number> <field1 value> <field 2> <field 3> There should be no need for nodes with more than 3 fields. If rules with more symbols on their RHS’s are present, such rules should be re-written as multiple rules satisfying the 3-symbol RHS maximum length constraint. These rules should be given numbers as if they were “real” grammar rules.