SYNTAX DIRECTED TRANSLATION

- Syntax Directed Definitions
  - A syntax-directed definition specifies the translation of a construct in terms of attributes associated with its syntactic components.
  - With each grammar symbol it associates a set of attributes, and with each production a set of "semantic rules" for computing values of the attributes associated with the symbols appearing in that production.
  - The grammar and the semantic rules constitute the syntax-directed definition.
- Translation Schemes
  - A "translation scheme" is a context-free grammar in which program fragments called "semantic actions" are embedded within the right sides of productions.
  - A translation scheme is like a syntax-directed definition, except that the order of evaluation of the semantic rules is explicitly shown.
- Syntax-Directed Translation
  - conceptually, with both syntax-directed definitions and translation schemes, we parse the input token stream, build the parse tree, and then traverse the tree as needed to evaluate the semantic rules at the parse tree nodes.
  - Evaluation of the semantic rules may generate code, save information in the symbol table, issue error messages, or perform any other activities.
  - Special cases of syntax-directed definitions can be implemented in a single pass by evaluating semantic rules during parsing, without explicitly constructing a parse tree.
  - An attribute can represent anything we choose: a number, a string, a type, a memory location, or whatever.

- The value of a "synthesized attribute" at a node is computed from the values of attributes at the children of that node in the parse tree.
- The value of an "inherited attribute" is computed from the values of attributes at the siblings and parent of that node.
- A parse tree showing the values of attributes at each node is called an "annotated parse tree".

• Example
- The syntax-directed definition given below is for a desk calculator program.
- This definition associates an integer-valued synthesized attribute called "val" with each of the nonterminals E, T, and F.
- The token "digit" has a synthesized attribute "lexval" whose value is assumed to be supplied by the lexical analyzer.

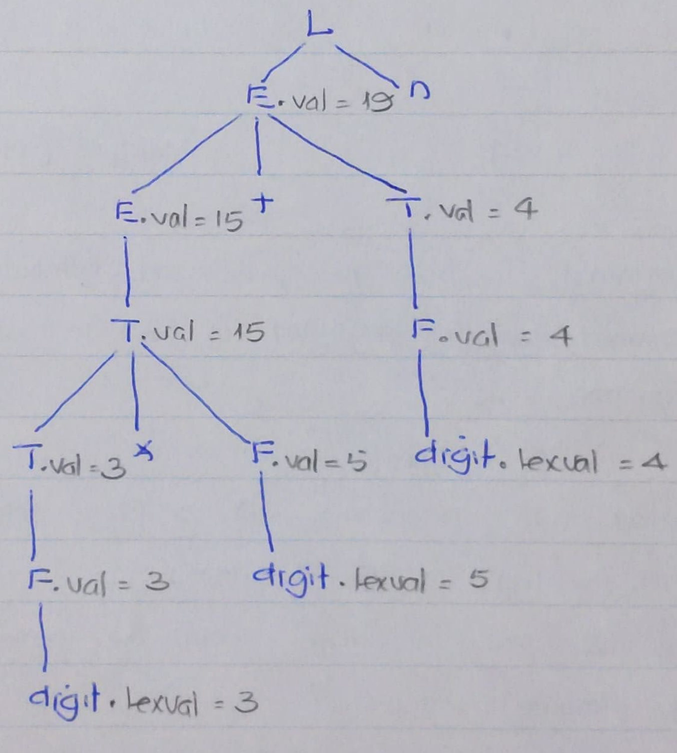
Production	Semantic Rules
$L \rightarrow E n$	print (E.val)
$E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
$E \rightarrow T$	$E.val = T.val$
$T \rightarrow T_1 * F$	$T.val = T_1.val * F.val$
$T \rightarrow F$	$T.val = F.val$
$F \rightarrow (E)$	$F.val = E.val$
$F \rightarrow digit$	$F.val = digit.lexval$

- In a syntax-directed definition, terminals are assumed to have synthesized attributes only, as the definition does not provide any semantic rules for terminals.
- Values for attributes of terminals are usually supplied by the lexical analyzer.

- Synthesized Attributes
  - A syntax-directed definition that uses synthesized attributes exclusively is said to be an "S-attributed definition".
  - A parse tree for an S-attributed definition can always be annotated by evaluating the semantic rules for the attributes at each node bottom up, from the leaves to the root.
  - Example: The annotated parse tree for

    3 * 5 + 4 n

    for the syntax-directed definition given earlier is shown below:



- Inherited Attributes
  - An inherited attribute is one whose value at a node in a parse tree is defined in terms of attributes at the parent and/or siblings of that node.
  - Example: In this example, an inherited attribute distributes type information to various identifiers in a declaration.

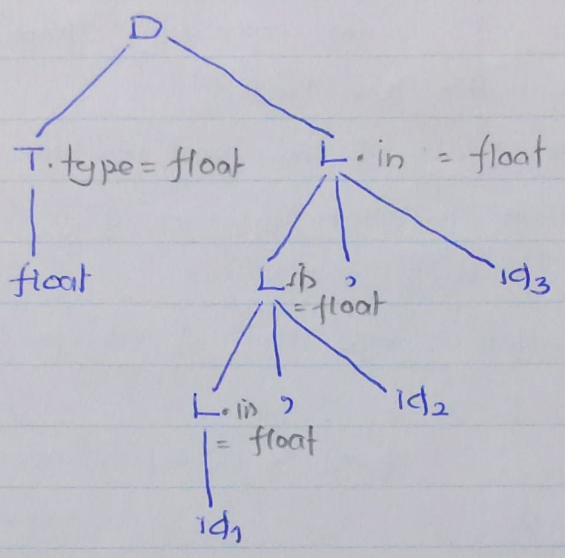
Production	Semantic Rules
$D \rightarrow TL$	$L.in = T.type$
$T \rightarrow int$	$T.type = integer$
$T \rightarrow float$	$T.type = float$
$L \rightarrow L_1, id$	$L_1.in = L.in$
	addtype (id.entry, L.in)
$L \rightarrow id$	addtype (id.entry, L.in)

  - The nonterminal T has a synthesized attribute "type", whose value is determined by the keyword in the declaration
  - The semantic rule

    $L.in = T.type$

    associated with production $D \rightarrow TL$, sets inherited attribute L.in to the type in the declaration.
  - The rules then pass this type down the parse tree using the inherited attribute L.in.
  - Rules associated with the productions for L call procedure "addtype" to add the type of each identifier to its entry in the symbol table (pointed to by attribute 'entry').

- Figure below shows an annoted parse tree for the sentence float id1, id2, id3



- At each L-node, we call the procedure "addtype" to insert into the symbol table the fact that the identifier at the right has type 'float'.

• Dependency Graphs
  - If an attribute 'b' at a node in a parse tree depends on attribute 'c', then the semantic rule for 'b' at that node must be evaluated after the semantic rule that defines 'c'.
  - The interdependencies among the inherited and synthesized attributes at the nodes in a parse tree can be depicted by a directed graph called a 'dependency graph'.
  - Before constructing a dependency graph for a parse tree, we put each semantic rule into the form $b = f(c_1, c_2, \ldots, c_k)$, by introducing a dummy synthesized attribute 'b' for each semantic rule that consists of a procedure call.
  - The graph has a node for each attribute and an edge to the node for 'b' from the node for 'c' if 'b' depends on attribute 'c'.