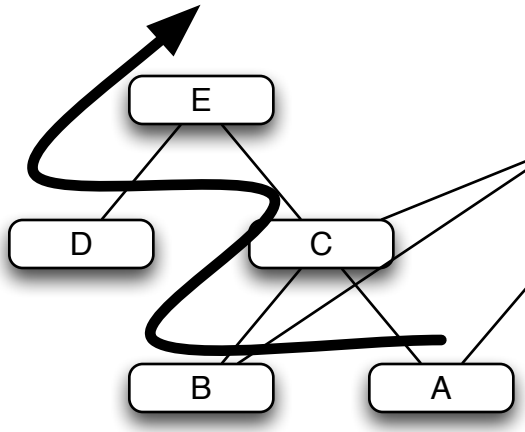
Hints on code generation

❶ Some overall architectural choices you will need to decide on include whether you will use an intermediate language (e.g., stack machine code) and your basic strategies for memory storage (literals, variables and temporary results).

❷ Traverse the tree from right-to-left and from the bottom up, i.e., visit the right sub-tree first, then the left, then the parent. (If there is only one child or no children, either visit the child first or just visit the node itself, respectively.)

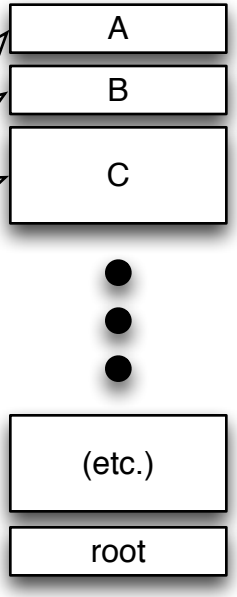


Expression tree

❸ As you visit each node, output a chunk of code based on:
- a template for the kind of node it is;
- instantiated with parameters based on the content of the node itself.

For example, for a literal you might generate a SET command, a DATA/COPY sequence or a DATA/LOAD sequence, depending on the actual numeric value involved. (Other node types involve similar but often more complex choices.)

❹ The "chunks" of code which you generate need to be held while you process them: you might keep them in an array, but a list or vector would be better. (A raw String is also possible, but makes any later analysis very difficult.)

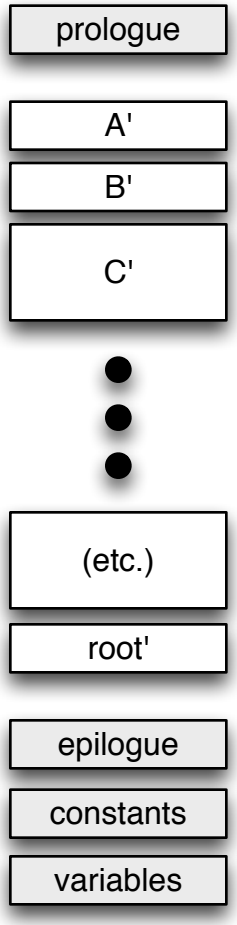


Intermediate code

"x"	R5
"y"	#VARY
"12"	...

❺ In addition to you code sequence, you may generate (and use) information about variables & constants along the way: you can store this information in a hash table. (You also might not need the table at all, depending on your storage strategies.)

Final PC-231 code



❻ Whether or not you use a special intermediate language, you will probably need to generate some special "chunks" of PC-231 code at the beginning and end of the program to set up pointers, write out the final result, etc.