Write a definition table module for your MINILAX compiler, following the design discussed in class. Define a set of connection routines that will allow you to compute the attributes and verify all conditions based upon them without building a structure tree for the program. This means that you may need to define more than one connection per abstract syntax rule: The recursive descent parser is effectively making one depth-first, left-to-right traversal of the structure tree and you may need to evaluate attributes on each visit to a node representing and abstract syntax rule. For example, the visit sequence for rule A_2_3_1 requires some attribute evaluations before the visit to the first child, some between the visits to the children, and some after the visit to the second child. You may find it useful to build a skeleton for your connection routines by text-editing the visit sequences from optout.

You will not need explicit variables for any .env attributes or any of the generated attributes, because these are subsumed by the definition table module and the storage allocation module that you will build for the next homework assignment. Not all of the remaining attributes are needed for this assignment.

Directory “556/hw10drv” contains support software for the attribute storage of the compiler, a driver program and the interface specification for the definition table module. The files in this directory are:

- deftbl.h: Interface specification for the definition table module. The functionality of these routines is that discussed in class, although the names, arguments and results have been slightly changed to reflect the requirements of MINILAX and the conventions of C.

- aladin.h: A C definition of the ALADIN data types, functions and optimized storage needed to compute and check the type attributes. There is no guarantee that this file is complete — you may need to add variables or stacks to it.

- stkmac.h: Utility operations to implement the global stacks assumed by GAG.

- driver.c: The parser driver.

You may either use your parser or take the one given as the solution to Homework #6. There are several advantages to using your own code if you possibly can: you understand its failings (if any), you will be able to see how your early local design decisions stand up to expanding requirements, and you will get more satisfaction in the end from having done it all yourself. On the other hand, if your previous solutions have serious errors or limitations and you spend lots of time fixing or avoiding them then you will not
learn what you should from the current assignment.

The purposes of this assignment are to familiarize you with the details of the constant-time definition table and to illustrate the interaction between a recursive-descent parser and the remainder of a compiler. It is worth 20 points, and is due at the beginning of the lecture on April 1.