1) I would add a test to the NUL Tc ase in the scanner to check for “{” and scan for “}”, using the same strategy that is used for comments currently. An invocation of NOTCMT with argument “{” would remove that character from the comment continuation set, and then the standard loop would find a tab, newline or the end of the comment. The costs are the same as those for the current comment style in time, and in space we must pay for another comment routine. It would be possible to combine the central loops of the two processors into a single procedure, but this would probably not be worth the effort.

2) There is no upper limit on the length of the source line other than the absolute constraint on the amount of memory available for the input buffer. Remember that enough memory must be reserved for one line plus one source block. The line length cannot exceed one source block, but the length of a source block is determined by the space available rather than any property of the source file.

3) The limitation on the size of a line can be avoided by using getc, but there will still be memory limitations that restrict the size of a single token. In Minilax we never need to back up more than one character, so ungetc would suffice. Additional overhead on buffer length checking would slow down the scanner somewhat, but if the lexical analyzer was not the major bottleneck in the compilation this would not be a problem.

4) It is very useful to set predefined identifiers on initialization.

5) If a string is permitted to span lines, then it is impossible to determine its length before some part of it must be stored. This means that a commitment must be made to a memory area of a particular size for such a string. If the area proves to be too small, the string must be moved again. In fact, the string might have to be moved an arbitrary number of times. When a string is not permitted to span lines, its length can be determined and it can then be moved once to an internal storage area of an appropriate size.