is possible, perhaps with renaming, to simply unnest any nested procedures. However, if there are references to non-global, non-local variables, recoding is necessary. This situation can be corrected either by passing the variables as parameters or declaring them global, but each has its disadvantages.

A similar situation exists with the logical operators in Pascal and C. The logical operators in C are conditional; in an expression like \((a \&\& b)\), \(b\) is evaluated only if \(a\) is true. In Pascal, both \(a\) and \(b\) are always evaluated (at least in theory). This distinction is important only if \(b\) may have side-effects. On the assumption that this is rare, we ignore the distinction when translating from Pascal to C. A simple macro/function call could be used in C if it was crucial that both components be evaluated.

An example and comparison. Once the translator was complete, I tested it on a simple program. Number.p (see Figure 1) is a Pascal program that prepends a line number to the beginning of each line of its input as it is copied to its output. If number.p is compiled by the Berkeley Pascal compiler on a Sun 1.5 workstation running 4.1c Berkeley Unix, and executed on a file of 148,151 bytes and 5861 lines, it takes 66.3 seconds of CPU time and 25,745 bytes of memory.

Translating this mechanically to C with our Pascal-to-C translator, gives the file number.c (see Figure 2). Running this program on the same input, the C program took 132.0 seconds of CPU time and 22,444 bytes of memory. Thus, although we saved a little memory, the program took twice as long to execute!

This seems a little strange. Looking at the C program, the only coding that seems “strange” is the reading and writing of the characters within each line. The Pascal read(c) was translated into the equivalent fscanf(stdin, "%c", &c) in C. Similarly the write(c) in Pascal was translated to fprintf(stdout, "%c", c) in C. Reading and writing a single character is a common special case however, and C provides the more direct getc and putc for this special case. An easy pattern matching can change all such reads and writes of a single character into the getc and putc forms. A particular advantage of these special forms is that they are implemented by inline macro expansion. This substitution provides the modified number.c (Version 2) shown in Figure 3. This version of number.c took only 18.3 seconds and 18,052 bytes of memory.

Additional editing to bring the program into the more common flavor of C programs produces another version of number.c (Version 3) without the calls to the Pascal eof and eoln (see Figure 4). This