Reading: For next week, read Chapters 9 and 10. Actually, my lectures are out-of-sync with the weeks, so Chapter 9 may not start until Nov. 2. Also please read Chapter 15 on “Functional Programming Languages,” as we are beginning a thematic look at them and comparing ML to Lisp.

This set is for both hardcopy and on-line submission, the latter part in a file CSE305ps8NNN.sml. It refers to sample programs in the CSE305 directories on fork (which you should use in preference to yeager), under ~regn/cse305/LANGUAGES/*/LECREC07/, where * is the all-capitalized name of the language, here ML and C#. In addition to the Java 5 and Mono setups from previous assignments, you may need special setup to run g++ without getting a library-linking error when you try to run your program. My solution (used in lectures) is to create a one-line shell script I call gpp with body g++ -R/util/gnu/lib $1 $2 $3 (allowing for up to 2 other compiler switches, such as -O5 for high optimization level), but there may be better ways. To compile pointer code in C# you will also need to use the -unsafe switch with gmcs (talk about “PC” brainwashing:-).

The Mon.+Tue. 10am recitations next week again meet in the Commons Conference Room.

(1) Using the 'a btree datatype in the RecEx3.sml file (versions of which you can find in other files in /.../ML/LECREC07/ and other open ML directories), write a recursive function mirror which creates a mirror-image of a tree. (9 pts.)

(2) Write an ML generic tree datatype ('a,'b) abtree which has two kinds of binary nodes, Anode and Bnode. The 'b item component of a Bnode will be interpreted as (a string denoting) some arbitrary binary operator, but an Anode will strictly denote the assignment operator, nothing else. (The 'a item of a Leaf can be unspecified for now, but will eventually be a datatype whose branches include things like Lvalue of string || Rvalue of int, or mote general than that. Whether to include a separate Empty option is up to you—the pain of changing your mind for later versioning will be a major theme in the last weeks! I have also added to RecEx3.sml a routine for converting trees to strings so that the whole tree prints out.) Then write in ML the following routines (6+6 = 12 pts.):

(a) A version of the postorder function in RecEx3.sml for your new datatype.

(b) A version of mirror that mirrors the Bnodes, but not the Anodes.

The following is for hardcopy submission on Friday, Nov. 2, in-class. It bears some relation to problem (2), but not for end-content or grading purposes.

(3) Find the lines of “evil code” in the EvalOrder*.cc,java,cs files in /.../C#/LECREC07/ that pertain to the variables i,j,k1,k2. They are the same in each file.

(a) Translate them into our stack language. Plug in the given translations of i++ and ++j from the handout (which is now included in that directory too). (18 pts.)

(b) Then trace the execution of your stack code, noting the values of the variables after each line. (The tediousness of doing this by typing is why this is for-hardcopy. 12 pts.)

(c) Now compile and run the C++, Java, and C# code, without optimization, using both the Sun CC compiler and g++ for the C++ code. Which, if any, agree with the stack-language results? (6 pts.)
(d) Does `gmcs` change its answers when the `-optimize` (or `-optimize+`) flag is specified, as happens for `-O5` using `g++`? What does that suggest about the rules for evaluating C# expressions compared to C++ and Java? (12 pts. for brief essay answer).

(e) Now focus just on the first two “evil” lines, involving only i and j. Translate them into stack code again but using the mirror of the expression trees as in problem (2), before doing the left-to-right postorder transversal. Do the results agree with any of the above languages or compilers? (12 pts., for 60 on the problem and 81 on the set. Optionally, you may modify your ML code to check your mirrored tree, but this is not required. Again the two following lines with `i++` and `++j` are not included here, partly since I don’t know how their “mirror” would be defined.)