CSE305, Fall 2007 **Assignment 7** **Due Fri. Oct. 26**

Reading: For next week, read Chapters 7 and 8. Note that large parts of chapter 7 have already been covered. This set is for **on-line submission**, in files `CSE305ps7NNN.sml` and `ScanNNN.cs`.

The Mon.+Tue. 10am recitations next week will **finally** meet in the Commons Conference Room. The CSE Commons Conference Room is on the ground floor of the main brick UB Commons building, at the end of the spur in front that points north toward the Ellicott Complex. It has the wall nameplate "Truthing Lab." You can enter from the inside (go to the end of one long hall, then jag left-right and go to the end of the next hall) or from the outside door facing the bookstore parking lot—if both are locked, knock on the latter! (Recitation R2 meets in 260 Capen as-usual.)

(1) Write the following functions in ML. You must write them in hardcopy ("by hand" or printed, either is OK) *and* submit them on-line in a single file `ps6NNN.sml`, where `NNN` are your initials (in uppercase, please). Your submissions should include light comments expressing why your functions work correctly—in two cases the comments are needed for full credit.

(a) A function `sumSquares(n)` that returns the sum of all squares up to n, e.g. `sumSquares(4) = 1 + 4 + 9 + 16 = 30`. (6 pts.)

(b) A function `collatz(n)` that counts the number of times required to *iterate* the process `if n = 1 then stop else if n is even then n/2 else 3*n + 1` until you reach the value 1. For instance, `collatz(1) = 0` and `collatz(3) = 7` because the process $3 \longrightarrow 10 \longrightarrow 5 \longrightarrow 16 \longrightarrow 8 \longrightarrow 4 \longrightarrow 2 \longrightarrow 1$ has seven steps. A mathematician named Collatz conjectured that `collatz(n)` halts for all n, but no one has been able to prove it. To see the trickiness of the issue, run `collatz(27)` and optionally, make your function print out all the steps along the way. (9 pts.)

(c) A function `ascenders(L)` that weeds out elements one-at-a-time from an integer list if they are not greater than all previous elements. E.g. `ascenders ([2,6,3,4,5,8,3]) = [2,6,8]` (even though `[2,3,4,5,8]` would be a longer ascending chain), and `ascenders ([]) = []`. (Note: The new ML will type the argument as `int list` owing to your use of $<$ or $>$, and note that you must parenthesize `(ascenders(x::rest))` since `::` binds looser than function application.) (9 pts.)

(d) A function `listPref(L1,L2)` that checks whether list `L1` is an initial segment of list `L2`; i.e., they match until list `L1` is empty but `L2` may have arbitrary leftover elements. If `L1` has leftover elements when `L2` runs out, return `false`. (6 pts., for 30 total)

(2) Compile `sumSquares` into our stack language—it should work with `n` atop an initially empty stack, and should leave its value on an otherwise-empty stack. You may use a test-for-0 instruction. Include this as a (* comment *) in your submitted ML file for the function. (15 pts.) [As stipulated in Monday's lecture, you may use both/either the jump-if-zero and jump-if-nonzero forms of the instruction (call them `jif0` and `jifn0` respectively), and also `swap` for interchanging the top two elements on the stack, and/or `dup` to duplicate the top-of-stack element, as in the JVM. Jumping is to a label given with the instruction, which is copied to the instruction counter (IC) to effect the jump.]

(3) This question uses the file `Scan.cs` which was covered in recitations.

(a) Rewrite the `Scan` method using an old-style for-loop. In what respect is it more efficient? Comment on whether you feel it is more or less readable. (12 pts.)

(b) Suppose we knew of a particular "`T identityElement`" e for the binary function f being passed into the class, such that for all x of type `T`, $f(e, x) = x$. For example, the empty string is an identity element for `String` and the concatenation function, and 0 is an identity element for `BigInteger` and `Plus`. However, there is no identity for `BigInteger` and `Max`! Show how to use an identity element to improve `Scan` using the new for-loop syntax. (9 pts., for 21 total on the problem and 66 on the set)