CSE305, Spring 2007 Final Exam Wed. May 9, 2007

Open book, open notes, closed neighbors, 170 minutes. Do *all five* questions in the exam booklets provided. The exam totals 234 pts., subdivided as shown. *Show your work*, and explain your reasoning where it is called for—doing so may help for partial credit.

(1) (3+0+6+9+15+9+6+3+6+6 = 63 pts.)
Consider the following function written in ML

```
fun findDoubles(L) = case L of
    nil => nil
  | x::nil => nil
  | x::y::rest => if x = y then x::findDoubles(rest)
                           else findDoubles(y::rest)
;
```

(a) Calculate `findDoubles [1,3,3,3,5,5,2,2,2,2,6,5,5]`. (3 pts.)

(b) Suppose your original intent was to make a list `findRepeaters(L)` of all the repeating elements, so that you'd get `[3,5,2,5]` instead of your actual answer to (a). How would you modify the code? Wait—don't answer this yet. (0 pts.)

(c) Make a tail-recursive "helper function" `fdh(L,R)` that does the same thing as `findDoubles`, except that the output list is accumulated onto the given list `R`. (6 pts.)

(d) Now modify your `fdh` to answer part (b). Did using your answer to (c) make things easier than your first thoughts in (b), and if so, how? (9 pts.)

(e) Now write a predicate `findDoubles(L,R)` in Prolog, where `L` is given and the output is to be stored in `R`. You may use the built-in equality predicate `=`, but must use a cut in place of also using the inequality predicate `\=`. (15 pts.)

(f) Now write `findDoubles(L)` as a method of the `Array` class in Ruby. You may use `self` to refer to the array/list itself, and/or use the `at(i)` method. Note that for an array `arr`, `arr.at(i)` is a synonym for `arr[i]`. Note also that `arr.push(m)` and `arr << m` are also synonyms, with `m` going to the rear in each case. Return a new array—i.e., don't destroy the invoking array. (9 pts.)

(g) Explain in words why this code would be hard (indeed, annoying and kludgey) to write using a `foreach` loop on an array in C♯, or using the `.each` method in Ruby. (6 pts.)

(h) *Now* suppose you were coding `E findFirstDouble()` inside the generic linked-list class `LL<E>` from Assignment 4. As the name says, you're no longer returning a *list* of doubled items, but just the first such *item*. In case the invoking list had no doubles, what would you return? (You don't have to write any C♯ code, just answer the question. 3 pts.)

(i) Now in ML, you can use the built-in "option" datatype:

```
datatype 'a option = NONE | SOME of 'a
```

to make `findFirstDouble(L)` return `NONE` if `L` has no doubles, or `SOME d` if `d` is the first double. But why might this be annoying? Write the ML type signature of `findFirstDouble` as part of your answer. (6 pts.)

(j) Would coding `findFirstDouble(L)` be similarly annoying in Ruby? Explain why or why not. (6 pts., for 63 total)

(2) (0+12+9+6+9+12+6 = 54 pts.)

Consider the following expression, in the syntax of C/C++/Java/C♯. Note that it has an internal assignment—recall that assignment is treated as a binary expression operation in all of these languages.

```
z = (x - y + z) / (x = x + 2) * (y - 3*z)
```

And recall part of the standard BNF for expressions in these languages:

```
E1 ::=  Var = E1 | E
E  ::=  E + T | E - T | T
T  ::=  T*F | T/F | F
F  ::=  (E1) | Var | any_numeric_literal
Var::=  x | y | z | etc.
```

(a) Evaluate the expression when `x = 4`, `y = 6`, and `z = 2`. (Since this question is worth 0 pts., please **do not** raise your hand to ask a question about it!)

(b) Write a parse tree for the above expression in this grammar. You may abbreviate some productions, e.g. `F` can be a direct child of `E1`. (12 pts.)

(c) Now write an expression tree for the expression itself. (9 pts.)

(d) Use your tree in (c) to convert the expression into Postfix form. (6 pts.)

(e) Now compile the Postfix into our "rudimentary stack language." (9 pts.)

(f) Show the evaluation of your stack code in (e) when `x = 4`, `y = 6`, and `z = 2`. (12 pts.)

(g) Suppose `y` and `z` are marked `const` in a C++ program. Why might a C++ compiler produce code that gives a different answer from that in (f)? (6pts., for 54 on the problem)

(3) (12+9+6+6+12 = 45 pts.)

The program overleaf is written in a language called **Bury**, for "Ruby the wrong way." Like Ada it allows unlimited nesting of functions—the idea is to Bury your code deep inside lexical scopes ;-). The serious point of a made-up language here is that you don't know whether it uses static or dynamic scoping—indeed this could even be an option flag on the **Bury** compiler. The syntax of the program (overleaf) should be clear from your knowledge of Ruby and C♯ and Ada.

```
def Main()
    int k,x
    def A(int y) return int
        def B(int x) return int
            return x*(y + 1)
        end B
        def C(int y) return int
            int k = 2*y
            return B(k)
        end C
    /* Main body of A begins here */
    begin A
        if y = 1 then return C(3*y)
                 else return A(y - 1)
        end if
    end A
    def D(int x)
        Console.WriteLine("Answer is: " + (x + k))
    end D
/* Body of Main() begins here. */
begin Main
    k = 3
    x = A(k)
    D(x)
end Main
```

(a) For each of the four sub-programs `A,B,C,D`, draw up a table showing which of the three variable names `k`, `x`, and `y` are visible within that sub-program, and if so, which block it belongs to (i.e., was declared in). For example, the table for `Main` is: `k = Main.k`, `x = Main.x`, but no `y` is visible. (12 pts.)

(b) Trace out the allocation of stack frames during the execution of this program. Show the static and dynamic links from each frame. (Everything except the particular values of variables within the frames is independent of parts (c) or (d); you should be able to use this trace to help you solve (c) and (d) without having to re-draw it each time. 9 pts.)

(c) Suppose that Bury uses static scoping. Using the information in your answers to (a) and (b), carefully work out the value printed by this program. Show your work. (6 pts.)

(d) Now suppose that Bury is using dynamic scoping. Work out the value that would be printed now. (Showing where a value would change is enough for full credit. 6 pts.)

(e) The mantra in Bury is "Every function is a class, and every variable is a method." Following this mantra, sketch code in C♯ (or Java or C++) that defines a class hierarchy that mimics the structure of the above program. Use `M` for the base class and `A,B,C,D` as the names of the other classes, and give the methods `x()`, `y()`, and `k()` in the appropriate classes. You need not provide anything else—no constructors are needed, and you definitely should ignore the bodies of the functions in your class-hierarchy sketch. All we are interested in is the analogy between lexical scoping and inheritance. (12 pts., for 45 total)

(4) ($8 \times 3 = 24$ pts.)

True/False **without** *justifications*. Please write out the words `true` and `false` in full. No justifications are asked for—and be sure to leave at least a half-hour for the essay question next.

(a) Since the syntax of Ruby is based on the syntax of C, Ruby suffers from the "dangling-else" ambiguity.

(b) Upon execution of a pointer assignment `p = q`, the value of `p` is the address of the storage object `q`, so that `p` now points at `q`.

(c) In Ada, the loop variable `i` in the for-loop `for i in 1..15 loop ... end loop;` has value `i = 16` after the loop exits.

(d) In Java, the loop variable `i` in the for-loop `for (int i = 1; i <= 15; i++) {...}` has value `i = 16` after the loop exits.

(e) On executing a C♯ statement `Foo x = new Foo();`, where `Foo` is a class, a storage object is created on the stack as well as on the heap.

(f) One of the reasons ML is not object-oriented is that if you could do `class D < B` as with subclassing in Ruby, and you coded a function `f(L)` to which the ML compiler gave the type `B list -> B list`, then you would be unable to pass a `D list` object `L` to it.

(g) In Ruby, if you declare `class Swan < Animal` and `class Duck < Animal`, then it is not a compile-time error to add a `Duck` to a list of `Swan` objects. (You can give a two-word reason here.:-)

(h) In Standard ML, in a function call of the form `f(X,Y)` where `X` and `Y` stand for arbitrary expressions, both `X` and `Y` are evaluated before the body of `f` is executed.

(5) ($4 \times$ (3+3+3+3) $= 48$ pts.)

In the decade before Java came out in 1994, C++ was the pre-eminent OOP language. For each of the following four language features, state how Java 5 and/or C♯ on one hand, and Ruby on the other hand, have different features that work *in opposite directions* from C++. For each of (a,b,c,d), you must give one concrete related feature of Ruby and one of Java-or-C♯, one positive and one negative aspect of the Java/C♯ way of doing things, and one positive and one negative aspect of the Ruby way of doing things. The grading is based on concrete example features and your evaluations of them.

(a) C++ allows global functions, which in particular gives you freedom to add functions that handle any data type (though you can't edit source code within existing classes or `struct`s.)

(b) C++ is moderately strongly typed. In particular, you can freely mix ints and floats and doubles with each other, though you might lose some precision.

(c) Templates in C++ guarantee compile-time type safety (as opposed to casting from `void*`, or from `Object` in Java/C♯), but produce separate blocks of executable code for each client of the template class.

(d) C++ allows you to code classes with by-value semantics, and pointers to classes which use by-reference semantics.

End of Exam