

The **Final Exam** is on **Monday, Dec. 17** (the last day of exam period), **3:30–6:30pm** in **Knox 4**. It will be open-book, open-notes (i.e., “open-everything” except your neighbor). There will be a *Review Session* the week before, most likely on Friday 12/14, and I will give out a sample final exam next week. The exam will be “cumulative,” i.e., range over all parts of the course that were covered.

This assignment is for **on-line submission only**, in files `CSE305ps12NNN.rb`, `StackDriverNNN.cs`, `StackCodeNNN.rb`, and `CSE305ps12NNNrept.txt`, where as usual NNN are your initials. I have decided to mandate for the C# submission that the provided classes `StackCommand` and `StackExecutor` *not be modified*, and since also the `Lookup` class was never intended to be modified, the former two (and all the little non-client classes and subclasses) can go in a file `StackCode.cs` which should not be modified, and the latter in a file `Lookup.cs`. Thus the C# code now has the same file organization as the ML code. Note that C# does not have the “include” facility of C/C++, nor the directory-lookup conventions of Java, but what makes separating the files possible without involving details of C# namespaces and assemblies is that the Mono `gmcs` compiler accepts multiple files as arguments. If you do

```
gmcs StackDriverNNN.cs Lookup.cs StackCode.cs
```

then you will get `StackDriverNNN.exe`. Oddly (to me), the `.exe` file is always named for the first file given, not the one that has `Main`, even though `gmcs` has a command-line switch to specify which of multiple classes with `Main` is the one to enter. (If you have already downloaded `StackCode.cs` when it was all in one file, it is easy to split it into three files, the first part being `Lookup.cs` and everything else before the `CodeMaker` class being `StackCode.cs`.) *The Ruby program is being kept as one file*—though if you wish to split it into three files (with class `Mstack` in either `Lookup.rb` or `StackCode.rb`) and submit just `StackDriverNNN.rb` instead, that is also OK.

Submissions. The Ruby exercises (1)–(3) are due online by 5pm on Friday 12/6 “as usual,” in `CSE305ps12NNN.rb`. The remaining parts must have an *initial submission* by that date; we will test your code over the weekend and give feedback, and in case it needs to be revised/completed, the final due date and time is **5pm, Wed. 12/12**.

(1) Code the `sumSquares` function recursively in Ruby. (6 pts.)

(2) Code the `ascenders(A)` function in Ruby, where `A` is an array of `Integer`. Then code the same function for arrays of any type, using *value-equality* which (opposite to Java) is `==` in Ruby, and assuming that you have a passed-in function `lt` for doing strict less-than comparisons. For a comparison of equality operators, see

themindstorms.blogspot.com/2006/06/parallel-of-equality-in-2-worlds-ruby.html

Your code should print out both the original and final arrays. (9+15 = 24 pts.)

(3) Finally add your `ascenders` method to the `Array` class, simply by “re-opening” the class, and show it running with syntax such as `[1,1,2,3,3,1,5,2,2].ascenders()`, which should yield `[1,2,3,5]`. (See the file `fibtest.rb` for how “open classes” simply work. 9 pts., for 39 total on this part.

Mini-Project: “Designing For Extension”

(4) Add support for the following new stack-language commands, both of which are treated on the stack-language handout:

- `dup`, which duplicates the top stack element, whether value or reference.
- `postinc` and `preinc`, translating (e.g.) `x preinc` by `x x fetch 1 + store` (or `x dup fetch 1 + store`) as given on the handout. This should be done by creating `PreInc` and `PostInc` stack commands, but then invoking a method `recompile` on the `List<StackCommand>` code object *somewhere* in the code.

This will be programmed in C# and in Ruby. In C# the groundrule is that you may not modify the `StackCommand` and `StackExecutor` classes, and should revise `CodeMaker` in a manner that is not specific to the three new stack commands, so that you can more easily subclass it to add that functionality. You may put the new `CodeMaker`, your extension, the new subclasses of `StackCommand<K,V>`, and your revised `StackDriver` class with `Main` all in the `StackDriverNNN.cs` file. In the Ruby code there is no “CodeMaker” class (you can move the file-scope “main” code into one, if you like) and no such groundrule: it’s being viewed as a “script” all of which you own.

The role-playing motivation is that `StackExecutor` is viewed as a third-party library class, while `CodeMaker` is “second-party” code by another wing of the project team—you do not own it and it can’t be made specific to your add-on of support for three commands, but you can advise on how it should be “designed for extension.” In the Ruby code, however, there is no idea of a “final” unchangeable class—you can re-open classes and make your own tailored versions at will.

A specific “quality points” consideration in the C# code is that you should minimize “redundant code” between the `CodeMaker` class and its extension—i.e., revise `CodeMaker` so that extensions can conveniently re-use its code when handling stack-language commands that belong to the original data set. This is also part of “designing for extension.” Beyond that, the criteria are more general but should be familiar: write clear, concise, efficient code, with comments as appropriate.

Points: 60 pts. for the C# code, 45 for the Ruby code. One-third of the points are for submitting by 12/7 something that is good enough for us to test. Code that functions correctly by the 12/12 revision due date is guaranteed two-thirds credit; i.e. “quality points” are limited to one-third of the points. The following essay questions are 15 points, each, making 180 points total on the mini-project.

Essay Questions: (As posted on the newsgroup—no change)

1. Discuss the effect of programming with strong-typing/generics in C# versus “Duck-Typing”/any-object-parameter in Ruby. Did the C# compiler ever catch a bug for you that might have gone undetected in Ruby? How did the typing regime affect the time it took you to write the code?
2. Note that the Ruby code *does* check that the item stack has Values vs. References/Keys when it is supposed to, but in which language were you more tempted not to do such a check when translating `postinc` and `preinc`. As for `dup`, which applies to both the case where the top item is a Value or a Ref/Key, would you even be able to use the methods in the `StackCommand` base class which do the checks? Is bypassing them by calling `stack.Pop` directly (as you are welcome to do) really safe?
3. Compare/contrast the ML code’s having all of the `eval` cases together in one place in the code, versus the C# and Ruby code having “eval” methods in each subclass. What are advantages/disadvantages of each? Which style makes the code easier to *extend*—indeed, how would you extend the ML `eval` function at all? Note that ML pattern-matching uses a “closed world assumption,” assuming that there are no other cases, similar to how Prolog assumes that if it doesn’t have a positive case of a predicate, then the predicate is false. “Sketch” how you could try to extend the ML code via a `datatype newCommand = Old of StackCommand | New of ...` (with `Dup,PostInc,PreInc` as new branches), —but how would such *explicit tag-lookup* make the code “yucky”?
4. As a separate issue from item 3, is it a good idea put the responsibility for eval-ing a single command in a separate method from the one that iterates through a list of commands? I did this in the stack-execution module of all 3 languages, but perhaps in the C# `CodeMaker` class the code you’re given could better follow this advice? Describe how separating a similar responsibility [might have] helped you make “CodeMaker” more “re-usable.” (Now that I’ve mandated revising `CodeMaker.cs` and then extending it, I no longer ask you if you decided instead to repeat/revise `Tokenize` totally...)
5. Finally, does the ability to “re-open” a class in Ruby—such as I did with `String` above and you can do by e.g. re-opening the `StackCodeEx` class to put the `recompile()` method *there* (note that the Ruby code has no “CodeMaker” class to extend)—make the issues in item 4 unnecessary? But what could be a disadvantage of having multiple implicit versions of the `String` or `StackCodeEx` classes?