CSc 372, Fall 2006
Mid-Term Examination Solutions

TOP SECRET

Circumstances have still not allowed two students to take a make-up for this exam. Guard these solutions and your exam closely.

A word about grading

The best way to be sure your work is scored equitably is get a problem right—there's no chance of being given the wrong amount of partial credit. We try very hard to make equivalent deductions for equivalent errors but it's very hard, especially with a large class. In some cases specific deductions and/or extra credit awards are cited below. If a mark on your exam differs from one of those you should definitely let me know. If you find that you got a lower mark than a colleague for a similar error, let me know. In some cases it might be that your friend got lucky. If so, I won't change your friend's mark but I also won't compound the error by making the same wrong deduction for you.

If you believe that an answer for one of the code-centric questions is correct then before coming to see me you should get on a machine and see if it actually works. If it does, come and see me. If it doesn't work then figure out the smallest change that makes it work. If you still think the deduction is inequitable then come and see me.

Please double-check that the scores written on each problem match the scores on the cover sheet.

As the syllabus states, you're encouraged to contest any score that you don't think is equitable.

Problem 1: (5 points; average 4.61, median 5)

The instructor often says "In ML we never change anything; we only make new things." What does he mean by that?

Here are my favorite complete answers:

"Everything is immutable."—Qiyam Tung

"There are no variables in ML, only aliases to unchangeable pieces of data."—Garrett Hoxie

A common error was to have a variable-centric answer, perhaps saying simply that a `val` might hide an earlier binding of the same name. That was a 1.5 point deduction.

Problem 2: (5 points; average 4.36, median 5)

(a) Write an ML function `fun firstLast(L)` *that returns a tuple consisting of the first and last elements of the list* `L`.

```
fun last [x] = x
  | last (_::xs) = last(xs)

fun firstLast(x::xs) = (x, last(x::xs))
```

(b) What is the type of `firstLast`*?*

```
'a list -> 'a * 'a
```

Problem 3: (5 points; average 2.28, median 1.5)

Flatten one-element lists.

```
val f = map hd
```

I was surely disappointed by the results on this one. I wish more students had put down something that worked, even if it was longer than I requested. Some students even erased answers and left it blank. Remember that on my exams something is no worse than nothing—give me a chance to give you partial credit. A blank is a zero, period.

Problem 4: (6 points; average 4.67, median 6)

Write an ML function `eo(L)` *that returns a list consisting of every other element of the list* `L`*. (That is, the 2nd, 4th, 6th, 8th, ... elements.)*

```
fun eo(x::y::more) = y::(eo more)
 |  eo _ = []
```

Problem 5: (12 points; average 8.59, median 9)

Without writing a function that is directly or indirectly recursive*, write an ML function* `ints_to_string(L)` *that produces a string representation of* `L`*, an* `int list`*.*

```
fun ints_to_string [] = ""
 |  ints_to_string L =
    let
        val almost =
            concat(map (fn(s) => ", " ^ Int.toString(s)) L)
    in
        implode(List.drop(explode almost, 2))
    end
```

I ended up working through a fairly similar problem—`printN`—during the Q&A session so I tried to compensate for that by saying a lot about it in a follow-up note to the list on Monday night.

Common errors were a trailing `", "` (-3); a `tl` too few (-1); producing just the integers, no commas (-5); and doing little more than `map Int.toString` (-8).

Problem 6: (15 points; average 13.03, median 15)

Without writing a function that is directly or indirectly recursive*, write an ML function* `show_lists(L)` *of type* `(string * int list) list -> unit` *that prints the contents of* `L`*, prefixing each* `int list` *with the specified label.*

```
fun its' [] = "<empty>"
 |  its' L = ints_to_string L;

fun show_lists(L) =
    print(concat(map
         (fn(label, IL) => label ^ ": " ^ (its' IL) ^ "\n") L))
```

Many solutions for both problem 5 and problem 6 used folding instead of a `map` and `concat` (or folding with `op^`). I see that as good news and bad news: I think that shows great understanding of folding but always seek the simplest solution—it has the greatest chance of being correct.

We ended up grading this problem in a somewhat qualitative fashion. For the most part we classified solutions as "A", "B", "C", "D", or "F" and awarded 15, 12.5, 10, 5, or zero points, respectively. If you see something like "A-15" on yours, that's why.

A very common error was to return `unit list` instead of `unit`. We didn't deduct for that but did award a point of extra credit if `unit` was returned. (Be sure we gave it to you if that's what you did.)

Problem 7: (10 points; average 1.83, median 0)

Consider a folding that operates on an `int list` *with an even number of values, all greater than zero, and produces a list of pair-wise sums:* `[1st+2nd, 3rd+4th, ..., (N-1)th+Nth]`

A fair number of students solved this problem or got close enough to make me happy. Mr. Bressel's solution was surely the best. I'd like you to think more about this problem so I'm not going to publish a solution here but if you're dying to know, see me or get to know Mr. Stout, Mr. D. Nguyen, Mr. Ghigliotti, Mr. Shokirev, Mr. Patki, Mr. Sarando, Mr. Tung, Mr. Bressel, Mr. Sortelli, Mr. Wallis, or Mr. Nation. (alphabetical by login name)

Problem 8: (4 points; average 2.25, median 2)

(a) Is the following ML function declaration valid? If valid, what is the type of `f1`*? If not valid, explain why it is not valid.*

```
fun f1 () f2 () = [f2];
```

It's valid. The type is `unit -> 'a -> unit -> 'a list`

I posted a question very similar to this to the mailing list. Remember that I think of my postings to that list as part of the course material.

There were a few exceptions but if you put down any type we usually considered that as evidence that you viewed the declaration as being valid and gave you a point for that.

(b) Write an ML function `g` *whose type is*

```
int -> (int list * int) -> (string list) -> (bool * real)
```

```
fun f 1 ([1],1) [""] = (true,1.0);
```

Problem 9: (4 points; average 2.71, median 3.25)

(a) Using nothing but a `val` *binding and the composition operator, create an ML function* `f(s)` *that returns a copy of the string* `s` *with the first and last characters removed. Assume that* `size(s) >= 2`*.*

```
val f = implode o rev o tl o rev o tl o explode;
```

(b) As you've defined it, what is the type of `f`*? (Don't forget to properly account for the intermediate functions in the composition.)*

```
string -> string
```

A few students fell victim to the bum steer about the intermediate functions. I guess they overlooked slide 162 and missed the time or two in class when I specifically said to watch out for this on an exam.

Problem 10: (16 points; average 9.85, median 12)

Write a Ruby program that reads from standard input a script of interaction with `irb` and combines the expressions and results on a single line, vertically aligning the results, based on the longest input expression. Each combined line is followed by a blank line.

```
exprs = []
results = []

if ARGV.length != 0
    gap = ARGV[0].to_i
else
    gap = 1
end

while line = STDIN.gets
   exprs << line.chomp
   results << (STDIN.gets || (puts "Error: short input"; exit))
end

longest = exprs.max{|a,b| a.size <=> b.size }.size

for e in exprs
    printf("%s%s\n", e.ljust(longest) + (" " * gap), results.shift)
end
```

Common deductions: Doesn't handle command line option (-2); doesn't handle "short input" case (-2); doesn't bother at all with alignment (-4); gets alignment wrong (-2). A lot of solutions aligned columns based on the longest input line instead of only considering the expressions (-2).

Problem 11: (2 points; average 1.62, mean 2)

Write a Ruby program that reads all lines from standard input and prints them on standard output in reverse order, last line first, first line last. Assume there is at least one line and that the file ends with a newline.

For two points of extra credit, have less than 25 characters in your solution.

```
puts readlines.reverse
```

I figured this one would be a cream puff since (1) it was the first thing we wrote in Ruby (slide 9), (2) I posed an in-class challenge to do this, and (3) I wrote the above on the Elmo when discussing (2)!

Problem 12: (3 points; average 2.02, median 3)

Here is a line of code from a Ruby method:

```
line = (gets || return)
```

Imagining the reader to be a Java programmer with no knowledge whatsoever of Ruby, explain its operation in each of the possible cases that might arise when it executes. Ignore the open-command-line-arguments-as-files-and-read-them behavior of `gets`.

If `gets` produces a line then it is assigned to `line`. If not, the `return` terminates the current method.

Many students said mentioned something to the effect of "the value produced by `return` is then assigned to `line`". That resulted in a deduction of 1.5.

Problem 13: (8 points; average 5.48, median 6.25)

Write a Ruby iterator named `upto_limit(a, limit)`. *The argument* `a` *is an array of integers;* `limit` *is an integer.* `upto_limit` *yields the values of* `a` *in turn (*`a[0]`, `a[1]`, *...) continuing while the sum of the yielded values is less than or equal to* `limit`. `upto_limit` *returns* `a`.

```
def upto_limit(a, limit)
    sum = 0
    for i in a
        sum += i
        if sum <= limit then
            yield i
        end
    end
    a
end
```

Common errors: Doesn't return `a` (-1.5); yields wrong values due to logic error (-1.5), prints instead of yielding—a fundamental misunderstanding (-2.5).

If a solution simply yielded the values in `a` we only gave it two points but at the moment seems like too big of a deduction. If you got that 2, see me and I'll change it to 3.5.

Problem 14: (5 points; average 2.85, median 4)

Write a Ruby method `extract(s, m, n)` *that extracts a portion of a string that represents a hierarchical data structure. m is a major index and n is a minor index. Major sections of the string are delimited by slashes and are composed of minor sections separated by colons.*

```
def extract(s,m,n)
    majors = s.split("/")
    majors.delete("")
    major = (majors[m-1] || return)

    minors = major.split(":")
    minors.delete("")
    minors[n-1]
end
```

A hint specified "*Assume that* `"|20|1|300|".split("|")` *produces* `["20", "1", "300"]`. With that in mind the `delete("")` calls above are not needed.

Common deductions: Using `m` and `n` instead of `m-1` and `n-1` (-0.5); not checking whether the major reference is out of bounds (-0.5).

Extra Credit Section (one point each unless otherwise indicated; average 2.86, median 2; high 8)

(1) *Imagine that you have an ML function named* `f` *and a Ruby method named* `f`. *Does* `[f]` *produce an analogous result in both languages? If not, how do the results differ?*

The result is different. In ML it is a list containing the function `f`. In Ruby it is an array that contains the result of calling `f`.

(2) *For up to three points name three programming languages developed at the University of Arizona and give an example of a valid expression in each that involves an operator.*

See slide 14 in `intro.sli.pdf`. I believe that `3+4` is a valid expression in all those languages except perhaps SIL2.

We went ahead and gave a point for just the language name. If you cited a valid expression it was worth

another half-point per language.

(3) *For one point each, write* `curry` *and* `uncurry` *in ML.*

```
fun curry f x y = f (x,y)

fun uncurry f (x,y) = f x y
```

Very few got these right but quite a few got them reversed.

(4) *Assuming that* `s` *is a string, what is a Ruby expression that produces the same effect as* `s.dup` *but is both shorter and more difficult to type?*

`s+""` I believe that only one student, Mr. Tung, got this one.

(5) *What element of Ruby most closely corresponds to an anonymous function in ML?*

Blocks.

(6) *Simplify this Ruby expression:* `if a[0] == nil then false else true end`

The answer I had in mind was `a[0]` but that's a little inaccurate. Mr. Kovell and Mr. Maloney did better: `!!a[0]`. (Do you see the difference?)

I should have said "Greatly simplify..." but since I didn't I accepted any expression that produced the same result but that was simpler than the above.

(7) *Cite a contribution to knowledge made by Ralph Griswold.*

I hope that somebody will take the time to write a biography about Ralph. Here are a few thoughts I've got.

At Bell Labs Ralph led development of four SNOBOL languages. I believe he said SNOBOL took a day to implement. SNOBOL2 took a week. SNOBOL3 took a month. SNOBOL4 took a year. SNOBOL4 was very popular both in industry and academia. It was written with portability in mind and coded in a macro language called SIL (SNOBOL Implementation Language). It was ported to dozens of different machines. Quite a few books were written about SNOBOL4 by both him and others. For many years it was by far the most popular unconventional language to use for non-numeric problems on mainframe computers.

A language that almost nobody's heard of is SL5—SNOBOL Language 5. Ralph led development of it here at the University of Arizona and, as the name implies, it was a follow-on in the SNOBOL line. Ralph once said, "SL5 had everything—you could even change the procedure call mechanism". But in the end the team didn't like the overall result and the language was never released. It is fading into the sands of time.

I asked Ralph what his first ideas about Icon were and he said, "I was in the hospital [I don't know for what] pondering SL5 and I kept thinking that there must be something simpler." That turned out to be Icon. Great effort was taken to make Icon small (conceptually) but powerful. I remember once that Ralph patiently listened while I rattled off a list of features that I thought would be good in Icon. He said "Go ahead, but for every one feature you add, first find one to remove." He was exaggerating, but by only a tiny amount.

Perl guru Damian Conway was asked in an interview, "What languages other than Perl do you enjoy programming in?" He replied, "I'm very partial to Icon. It's so beautifully put together, so elegantly proportioned, almost like a Renaissance painting." In the `digibarn.com` poster cited in the intro slides Icon is described as "A general purpose language known for its beauty and grace."

Ralph founded this department and served as department head for many years. It's hard to imagine how difficult it is to simultaneously and successfully teach classes, conduct original research and publish about it, write successful grant proposals, build from scratch a department that is internationally recognized, direct and coordinate the work of several graduate students, and on top of all that, keep an academic department

running smoothly, bearing in mind that a collection of professors, each with their own priorities and often-strong opinions, are not easy to keep happy.

When Ralph "retired" he quickly rose to prominence in the weaving community by virtue of his interest and pursuit of mathematical problems in weaving.

Here's a quote about Ralph from one of his graduate students who is now a professor: "As one of the founders of the Bell Labs software culture which spawned UNIX, C, and many other essential contributions to modern software, Ralph Griswold brought to his academic research not only brilliance, but also experience and a value system that demanded that research ideas be tested by fire and proven useful and usable by real users, not just good-looking diagrams in academic papers."—Clint Jeffery

When I started working for Ralph as a graduate student I discovered that my was education was just beginning. He was probably the smartest guy I've ever known and surely the wisest, at least in this field. One of the great fortunes of my life was working for him.

Things that Ralph clearly saw as good ideas over thirty years ago, such as run-time type checking, garbage collection, and a well-designed collection of datatypes are now becoming widely accepted.

(8) (Up to five points.) Offer some intelligent observations about the applicability of type deduction, or something similar, in Ruby.

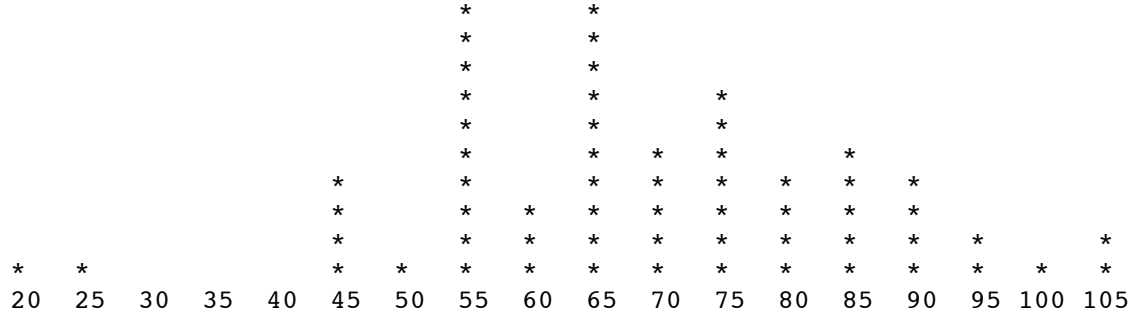
There has been quite a bit of research on various aspects of figuring out types in languages that don't use static typing, like Icon and Ruby. A fellow named Ken Walker wrote a type-inferencing Icon compiler about a decade ago for his dissertation. I believe he found that as a rule "type consistency" held in about 80% of the cases for the large body of Icon code that he tested with. You can read more here: `www.cs.arizona.edu/icon/ftp/doc/tr93_32.pdf`.

Ruby is more slippery than Icon however. For example, in Icon you can be sure that something like `x * 3.7` is always going to produce a floating point value. That's not so in Ruby. In Ruby you can't even be sure that `2+2` is `4`! However, a user who's interested in speeding up a Ruby program might be willing to sacrifice mutability for performance.

Overall results

The average on the exam was 68.9 and the median was 67.25. Here is the full set of scores and a histogram:

```
106.00, 104.00, 102.00,  96.50,  96.00,  91.50,  91.00,  90.50,  89.50,
 86.00,  85.00,  84.50,  84.50,  83.50,  82.00,  82.00,  81.00,  79.00,
 76.50,  75.50,  75.50,  75.50,  75.50,  74.50,  74.50,  71.50,  69.50,
 68.50,  67.50,  67.50,  67.00,  66.50,  66.50,  66.50,  66.00,  64.00,
 64.00,  63.50,  63.00,  62.50,  62.00,  60.50,  57.50,  57.00,  56.00,
 55.50,  55.50,  55.50,  55.50,  55.00,  53.50,  53.50,  53.50,  50.00,
 46.50,  45.25,  43.50,  42.50,  23.75,  18.50
```



I want to think about it a little bit more but I'm currently leaning against any sort of class-wide adjustment of the scores. If that were to happen I believe the maximum I'd add would be 6 points.