CSc 372, Fall 2006
Final Examination Solutions

Problem 1: (13 points; average: 10.8)

In this problem you are to write a Ruby program `xref.rb` that prints a cross-reference table of what identifiers appear on which lines in a Ruby program.

```
ids = Hash.new([])
lnum = 1
while line = gets
      line.scan($id_re).each {|id| ids[id] += [lnum] unless $kwds.include? id }
      lnum += 1
end

ids.sort.each {
      |pair|
      id = pair[0]
      lines = pair[1].uniq
      printf("%s: %s\n", id, lines * ", ")
      }
```

Problem 2: (8 points; average: 6.7)

The `connect` predicate on assignment 9 printed a representation of a sequence of cables. In this problem you are to write a Ruby method `parse_layout(s)` that parses such a representation and returns an array of arrays representing the cables.

```
>> parse_layout("M---MF-MF----M")
=> [["m", 3, "m"], ["f", 1, "m"], ["f", 4, "m"]]
```

Here's what I thought of right off the bat:

```
def parse_layout(s)
    r = []

    s.downcase!
    while s =~ /([mf])(-+)([mf])/
        r << [$1, $2.size, $3]
        s[$&] = ""
    end

    r
end
```

Mr. Ghigliotti and Mr. Sortelli used an approach like this:

```
def parse_layout(s)
    s.downcase.scan(/([mf])(-+)([mf])/).map {|e1,ds,e2| [e1,ds.size,e2] }
end
```

Problem 3: (2 points; average: 1.1)

Specify the contents of a Ruby source file, `tptnf.rb`, such that after loading it, ***2+2 is not 4***.

```
class Fixnum
    def + rhs
        0
    end
end
```

Common errors were to omit `class Fixnum` and to specify more than one parameter for the + method. (Remember that `a+b` is like `a.+(b)`).

Problem 4: (4 points; average: 2.7)

(a) In a non-recursive way, implement `between(+Low, +High, -Value).`

```
between(Low, High, Value) :- numlist(Low, High, List), member(Value, List).
```

(b) In a non-recursive way, implement `numlist(+Low, +High, -Value).`

```
numlist(Low, High, List) :- findall(Value, between(Low, High, Value), List).
```

The answers for `between` were mostly correct but only about a dozen answers for `numlist` were correct.

Problem 5: (6 points; average: 4.1)

Write a Prolog predicate `idpfx(+List, -Prefix)` that instantiates `Prefix` to the longest prefix of `List` such that all elements of the prefix are identical.

```
idpfx(L,P) :- reverse(L,R), append(_, P, R), allsame(P), !.
```

Problem 6: (14 points; average: 12.9)

Write a predicate `show_cost(C)` that prints a description and cost of the cable `C`. Cables are represented using a structure with the functor `cable`. The structure `cable(m,2,f)` represents a 2-foot cable with a male connector on one end and a female connector on the other.

```
show_cost(Cable) :-
      order(Cable, cable(E1, Len, E2)),
      map(E1,E1nm),
      map(E2,E2nm),
      cost(E1nm, E1Cost),
      cost(E2nm, E2Cost),
      cost(foot, FootCost),
      Cost is FootCost * Len + E1Cost + E2Cost,
      format('~w-foot ~w to ~w: $~2f~n', [Len, E1nm, E2nm, Cost]).

order(cable(E1, Len, f), cable(f, Len, E1)).
order(X, X).

map(m, male).
map(f, female).
```

Problem 7: (7 points; average: 8.1)

Write a predicate `halves(+L,-H1,-H2)` that instantiates `H1` and `H2` to the first and second halves of the list `L`, respectively. `halves` fails if `L` has an odd number of elements.

```
halves(L,H1,H2) :- append(H1,H2,L), length(H1,Len), length(H2,Len).
```

When I wrote my first solution for this I checked for an even number and then used `length` to form a half-sized list. I then realized that a description of halving, "find a place to cut this so you get two equal parts", works in Prolog, too!

Problem 8: (8 points; average: 3.3)

Write a predicate `wseq(+Words,-Seq)` that finds a sequence of the atoms in `Words` such that the last character of each atom is the same as the first character of the next atom. Here is a simple example:

```
wseq([W], [W]).

wseq(Ws, [W1,W2|Sorted]) :-
    select(W1, Ws, Rest),
    wseq(Rest,[W2|Sorted]),
    pair(W1,W2).

pair(A1,A2) :-
    atom_chars(A1,C1),
    atom_chars(A2,[C|_]),
    last(C1,C).
```

Problem 9: (2 points each; 16 points total; average: 12.1)

The expression `(f 1 2)` has meaning in both ML and Emacs Lisp. In Emacs Lisp it means to call the function `f` with the arguments `1` and `2`. Does it mean the same thing in ML? If not, what does it mean?

> In ML it means `((f 1) 2)` — call `f` with the argument 1. Then call the function produced by `(f 1)` with the argument `2`.

It's well known that thinking up names for variables and functions is not always easy and that bad names make code harder to understand. This is sometimes called the "naming problem". It can be said that with respect to Java, one of our three primary languages makes the naming problem worse. Another of the three lessens the naming problem. The third is roughly neutral—it requires about as much naming as Java. Which language is which? Briefly state why.

> I'd say that Prolog makes the naming problem worse because there's no nesting of expressions; ML makes things better thanks to partial applications and anonymous; Ruby is neutral.

With the Icon programming language in mind, what's meant by the term "failure"? Show two distinct examples of expressions that can fail or succeed depending on the values of the variables involved.

> When an expression is evaluated in Icon it either succeeds and produces a result or it fails and produces no result. Failure propagates to enclosing expressions and they fail in turn.
>
> Two examples of expressions that can fail are `x < y` and `s[n]`.

Prolog has predicates for comparison like `>/2` and `==/2` but it does not have predicates for arithmetic like, `+/2` and `*/2`. Why is that?

> Success or failure itself is a meaningful result that is manifested by control continuing through the goal (or not). Arithmetic expressions need to produce a result but an expression like `3 + 4` simply doesn't provide a place to specify a result. Arithmetic predicates would need to look like `add(3,4,R)` and things would get clumsy fast.

Which does Prolog use—compile-time type checking or run-time type checking? Or does the notion of type-checking not really apply to Prolog? Support your answer with a brief argument.

A number of students said that Prolog has no type checking—type mismatches simply produce "No" but that's incorrect. Consider `X is 3 + abc`. There's no error when that code is loaded but there is an error when that goal is evaluated.

Ruby's designer elected to have `string[n]` produce a integer character code instead of a one-character string. Ignoring possible performance considerations write a brief argument either in favor of this design decision or against it.

I personally think this is a terrible idea but that's perhaps because I'm used to Icon, which produces a one-character string for `string[n]`.

Write a simple Ruby method that takes advantage of duck typing and briefly explain how duck typing allows the code to be simpler, or more expressive, or etc.

The method

```
def double x
    return x + x
end
```

"doubles" `x` by adding it to itself. This has meaning for any type that has provided a meaning for addition.

The term "duck typing" appears to have originated in the Ruby community but the idea has been around for years; routines like `double` are said to be "polymorphic".

What is meant by the term "syntactic sugar"?

A syntactic construct that is not necessary but provides a more convenient way to express something.

Problem 10: (1 point each; 4 points total; average: 2.5)

Characterize each statement below as true or false.

__F__ *The instructor prefers the term "scripting language" to categorize languages like Icon, Perl, Python, and Ruby.*

It's true that I made a practice of calling Ruby a "scripting language" but I also said that in general I don't like that term to describe languages like Icon, Perl, Python, and Ruby. The industry may well be stuck with "scripting language" by now but an alternative I prefer is "agile language". However, when you start trying to characterize agile languages you often end up with only one thing in common: dynamic typing, so maybe "dynamically typed language" is the best term.

Lisp is an interesting case: it's dynamically typed but I think its syntax is sufficiently cumbersome to make me hesitate saying it's agile.

I think the term "scripting language" should be reserved for languages that let one orchestrate a series of commands in another computational domain with little interaction between the domains and/or little knowledge of either domain in the other domain. For example, most shells support scripts that string together commands but the only communication between the two domains is via command line arguments, termination codes, and standard input/output. The shell has no knowledge of what `cat`, `ls`, `find`, etc. are doing and the programs have no knowledge of the shell—they simply take arguments, maybe handle standard input/output, and produce an exit code.

__F__ *In Icon, the expression `write(1 to 10)` prints the numbers from 1 through 10.*

Just about everybody missed this one. When `write(1 to 10)` is evaluated the generator `1 to 10` produces 1,

the first value in its result sequence, and that is written out. To get all the numbers you need to put in a context where the generator is resumed until it is exhausted, like `every write(1 to 10)` or `write(1 to 10) & p()`, where `p` is a procedure that always fails.

F *The Prolog fact `p([A,B,C])` indicates that `p(X)` is true iff `X` is a three-element list whose values are all different.*

The use of three different variables allows unification with three different values but it does not require them to be different.

T *The regular expression `/[a-z][x][123]/` can be written more concisely.*

`/[a-z]x[123]/` (A one-character character class is equivalent to a single character.)

Problem 11: (18 points)

The average on the essay questions was 16.53.

Problem 12: (6 points, EXTRA CREDIT; average: 1.74)

Using the parsing idiom supported by Prolog's rule notation, write a predicate `parse(S)` that parses a simple Ruby string subscripting expression and outputs a line representing a method call that performs the same computation.

```
?- parse('s[1]').
s.charAt(1)
Yes

?- parse('line[10,20]').
line.substr(10,20)
Yes


parse(S) :- atom_chars(S,C), subexpr(R, C, []), show(R).

subexpr(sub(Id,D)) --> id(Id), lbrack, digits(D), rbrack.
subexpr(sub(Id,D1,D2)) --> id(Id), lbrack, digits(D1), [','], digits(D2), rbrack.

show(sub(Id,D)) :- format('~w.charAt(~w)~n', [Id, D]).
show(sub(Id,D1,D2)) :- format('~w.substr(~w,~w)~n', [Id, D1, D2]).

lbrack --> ['['].
rbrack --> [']'].
```

In many answers the `format` calls were in the grammar rules. There was no deduction for that but it's not entirely correct. The problem with printing in the rules is that there may be trailing characters, like `'s[1]x'`. If so the printing is premature.

Extra Credit Section (one half-point each unless otherwise indicated; average: 4.0)

NOTE: In general, incorrect but humorous answers worked, too.

(1) What programming language in the SNOBOL/Icon family was developed between SNOBOL4 and Icon?

Lots of students said "SNOBOL5" but it was "SL5" (SNOBOL Language 5).

(2) Name a programming language that allegedly supports more than seven (7) programming paradigms.

I made the briefest mention of it while looking at the Wikipedia "Multi-Paradigm" page during the last class but Oz is claimed to support eight programming paradigms. It can also be argued that due to its flexibility Lisp can support any paradigm so I accepted Lisp as an answer as well.

(3) Order these languages by age, oldest to youngest: Java, Icon, Lisp, ML, Ruby, Scala.

I'd say the order is Lisp, ML, Icon, Ruby, Java, Scala but Java and Ruby are very close. As long as those two were consecutive either order was acceptable for them. I counted an answer as correct if at most a single deletion yielded a valid ordering.

(4) (2 points) Write an ML function `eq(L1,L2)` of type `'a list * 'a list -> bool` that returns `true` iff the lists `L1` and `L2` are equal. Be sure to accommodate lists that contain lists.

It wasn't very conservation-minded to allow so much space for such a short answer but here's all you need:

```
fun eq(L1,L2) = L1 = L2
```

I should have deducted a little bit for `fun eq(L1,L2) = if L1 = L2 then true else false`, but I looked the other way.

(5) Imagining that Ruby has Icon's notion of failure, rewrite the following code to take advantage of failure:

```
for i in 0...len
  c = self[start+i]
    if c then
        r += c.chr
    end
end
```

There would be no need to see if `c` was `nil`:

```
for i in 0...len
  r += self[start+i].chr
end
```

(6) What is a connection of sorts between Java and constraint programming?

James Gosling's Ph.D. dissertation concerned the algebraic manipulation of constraints.

(7) In three words or less, describe Icon in terms of the languages we studied this semester.

What I had in mind was "Ruby meets Prolog" but other answers worked, too.

(8) In three words or less, describe Scala in terms of the languages we studied this semester.

What I had in mind was "Java meets ML".

(9) The names Java, Ruby and Icon are not acronyms. Create a humorous acronym for each that is somehow related to the language, like Lisp: Lost In Stupid Parentheses. (1/2 point each)

I was surprised by the number of good ones that turned up. I wish I had time to quote them all.

(10) (1 point) In SWI-Prolog the query `?- X.` produces an "Easter Egg" that alludes to a well-known book. What is the title of that book?

tHHGttG

(11) Who was the central character in the short film Jarwars Episode III, Revenge of the <T>?

Duke Vader

(12) According to assigned reading, the only well-known scholarly paper published by Bill Gates concerned what problem?

I believe that just about the only reading I assigned (rather than only suggested) was in the assignment 9 write-up: "There's even a Wikipedia article about pancake sorting. (Read it!)"

(13) On every lecture day the instructor ate breakfast at Millie's Pancake House. Estimate the total number of pancakes he consumed at Millie's during those breakfasts.

Zero. I prefer their waffles! I'll sometimes have the German Pancake but there was never enough time for one of those on lecture days.

Only Mr. Hoskins got this one, speculating that I was probably too busy flipping them to eat any.

(14) (2 points) In any language you wish, write a program to read this exam as plain text on standard input and output the total number of points for all the regular problems, i.e., don't worry about this extra credit section.

```
sum = 0
while line = gets
    if line.chomp! =~ /(\d+) points( total)?\)$/ then
        sum += $1.to_i
    end
end
puts sum
```

Mr. Merrill's solution was unique: The language he chose was English!

Overall statistics

The average on the exam was 86.57; the median was 85.5.

Here is the full set of scores:

```
110.00, 109.00, 108.00, 105.00, 104.50, 104.00, 103.00, 103.00, 102.00, 101.50,
99.00, 98.50, 98.00, 98.00, 97.00, 96.00, 95.00, 95.00, 94.50, 93.50, 92.00,
90.00, 90.00, 89.50, 87.50, 87.00, 87.00, 85.50, 85.50, 85.50, 84.50, 84.50,
83.50, 83.00, 82.50, 82.00, 82.00, 81.50, 81.00, 79.50, 78.50, 77.00, 76.00,
75.00, 75.00, 73.50, 73.50, 72.00, 70.50, 70.50, 69.50, 64.00, 64.00, 58.00,
46.50
```

Here is a histogram: