CS login: _____ Seat Number: _____

CSc 372 Final Examination
December 14, 2006

READ THIS FIRST

Read this page now but do not turn this page until you are directed to do so. Go ahead and fill in your login and seat number.

This is a 105-minute exam with a total of 100 points of regular questions and an extra credit section.

You are allowed no reference materials whatsoever.

If you run out of room, write on the back of a page. DO NOT use sheets of your own paper.

If you have a question, raise your hand. One of us will come to you. DO NOT leave your seat!

If you have a question that can be safely resolved with a minor assumption, state the assumption and proceed. Examples:

> Assuming `select(?Elem, ?List, ?Remaining)`
> Assuming `String#downcase!` imperatively converts all capitals to lower case.

BE CAREFUL with assumptions that dramatically change the difficulty of a problem. If in doubt, ask a question.

Unless explicitly prohibited on a problem you may use helper functions/methods.

Don't waste time by creating solutions that are more general, more efficient, etc. than required. Take full advantage of whatever assumptions are stated.

As a broad rule when grading, we consider whether it would be likely if the error would be easily found and fixed if one were able to run it. For example, something like `i + x` instead of `i + x.to_i`, or forgetting a `chomp` will be typically a minor deduction at worst. On the other hand, an error that possibly shows a fundamental misunderstanding, such as a `yield` with no argument for a block that expects one, will often lead to a large deduction.

Feel free to use abbreviated notation such as I often use when writing on the Elmo. For example, you might use a ditto instead of writing out the function name for each case or abbreviate a function/method name to `a_b_c` or `ABC`. Don't worry about matching parentheses at the end of a line—just write plenty and we'll know what you mean.

You need not include any explanation in an answer if you are confident it is correct. However, if an answer is incorrect, any accompanying explanation may help you earn partial credit.

If you're stuck on a problem, please ask for a hint. Try to avoid leaving a problem completely blank—that will certainly earn no credit.

It is better to put forth a solution that violates stated restrictions than to leave it blank—a solution with violations may still be worth partial credit.

When told to begin, double-check that your name is at the top of this page, and then **put your initials, or some other distinctive mark, in the lower right hand corner of each page**.

BE SURE to check that you have all 15 pages.

When you have completed the exam, enter your name on the exam sign-out log and then hand your exam to the instructor.

Problem 1: (13 points)

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. Here is a source file:

```
% cat -n map.rb
     1  def map(a)
     2      map_result = []
     3      for x in a do
     4          block_result = yield x
     5          map_result << block_result
     6      end
     7      return map_result
     8  end
```

NOTE THAT `cat-n` is being used to show line numbers. The file itself does not include those numbers.

Here is what `xref` does with `map.rb`:

```
% ruby xref.rb < map.rb
a: 1, 3
block_result: 4, 5
map: 1
map_result: 2, 5, 7
x: 3, 4
```

We see that the identifier `a` appears on lines 1 and 3. `block_result` appears on lines 4 and 5, etc. A given line number will appear only once for an identifier, no matter how many times the identifier appears on a line.

Here are some simplifications:

Assume that `$id_re` is a regular expression that matches identifiers. You might use it with `String#scan`:

```
>> "def map(a)".scan($id_re)
=> ["def", "map", "a"]
```

Assume that `$kwds` is an array of identifiers to ignore, like `$kwds = ["def", "end" ...]`

Recall that sorting a hash produces a list of two-element lists that are key/value pairs ordered by the keys:

```
>> h = {"a", 10, "b", 20}
=> {"a"=>10, "b"=>20}

>> h.sort
=> [["a", 10], ["b", 20]]
```

`Array#uniq` returns a copy of the array with all duplicates removed.

(Space for solution for problem 1.)

Data point: the instructor's solution is 13 lines in length.

```
% cat -n map.rb
    1  def map(a)
    2      map_result = [ ]
    3      for x in a do
    4          block_result = yield x
    5          map_result << block_result
    6      end
    7      return map_result
    8  end
% ruby xref.rb < map.rb
a: 1, 3
block_result: 4, 5
map: 1
map_result: 2, 5, 7
x: 3, 4
```

Problem 2: (8 points)

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"]]

>> parse_layout("F-------F")
=> [["f", 7, "f"]]
```

Assume that the input is well-formed, that there will always be at least one cable, and that all cables will be at least one unit long.

Problem 3: (2 points)

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

```
>> 2 + 2 == 4
=> true

>> load "tptnf.rb"
=> true

>> 2 + 2 == 4
=> false
```

Hint: Don't make this a hard problem. If your solution exhibits the above behavior it will be considered correct—behavior for all other cases is of no concern.

Problem 4: (4 points)

The built-in Prolog predicate `between(+Low, +High, -Value)` instantiates `Value` to each integer between `Low` and `High`, inclusive. The built-in predicate `numlist(+Low, +High, -List)` instantiates `List` to a list of the integers between `Low` and `High`, inclusive.

(a) In a non-recursive way, implement `between(+Low, +High, -Value).` You may use any predicates you wish, except for `between/3`.

(b) In a non-recursive way, implement `numlist(+Low, +High, -Value).` You may use any predicates you wish, except for `numlist/3`.

Incidentally, another way to think about this pair of predicates is this: (a) Using `numlist`, implement `between`. (b) Using `between`, implement `numlist`.

Problem 5: (6 points)

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. Assume that `List` has at least one element. `idpfx` always produces exactly one result. Examples:

```
?- idpfx([3,3,1,5],P).
P = [3, 3]

?- idpfx([1,2,3],P).
P = [1]

?- idpfx([3,3,3,5,3],P).
P = [3, 3, 3] ;
No
```

Hint: Recall that the first result of the query `append(A,B,[1,2])` is `A = [],B = [1, 2].`

You may use the predicate `allsame(L)` from the slides, which succeeds iff all values in `L` are identical.

Problem 6: (14 points)

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.

A set of `cost` facts represents the cost of the various components. For example, the facts

```
cost(male, 2.0).  cost(female, 3.0).  cost(foot, 0.50).
```

indicate that male connectors are $2.00, females are $3.00 and each foot of cable costs 50 cents. The cost of a cable is the sum of the cost of its components.

Here is a call to `show_cost`:

```
?- show_cost(cable(m,3,f)).
3-foot female to male: $6.50
Yes
```

IMPORTANT: If a cable has both a male and female connector, the output produced by `show_cost` ALWAYS describes the cable as "`...female to male...`", i.e., "female" first. (A cable is NEVER shown as "`...male to female...`"!)

IMPORTANT: Note that although "`m`" and "`f`" are used in the `cable` structure, "`male`" and "`female`" are output in the description.

Don't worry about any sort of error checking.

Problem 7: (7 points)

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` produces at most one result. Examples:

```
?- halves([1,2,3,4], H1, H2).
H1 = [1, 2]
H2 = [3, 4]

?- halves([1,2,3], H1, H2).
No

?- halves([],H1,H2).
H1 = []
H2 = []
```

For three points of extra credit, use no more than three *goals* to implement halves. (Goals, not clauses!)

Problem 8: (8 points)

NOTE: This problem is far harder than the eight points it is worth. It my be wise to save it for last.

RESTRICTION: You must base your solution on the *"pick one (with `select/3`), try it, solve with what's left"* idiom shown in the slides with brick laying and also used in the instructor's solution for `connect` on assignment 9. In particular, YOU MAY NOT USE the built-in `permutation/2` predicate or a similar predicate that you write yourself.

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([pop, up],S).
S = [up, pop]
```

The sequence is valid because `"up"` ends in the same letter, `"p"`, that `"pop"` starts with.

Here is a longer example:

```
?- wseq([slowly,the,apples,test,extra],Seq).
Seq = [test, the, extra, apples, slowly]
```

`wseq` produces all valid sequences:

```
?- wseq([tic,cat],S).
S = [tic, cat] ;
S = [cat, tic] ;
No
```

`wseq` fails if there is no valid sequence:

```
?- wseq([tic,tac],Seq).
No
```

Assume that there is at least one word—`wseq` always succeeds in that case—and that each word has at least one character.

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

Answer the following questions. A sentence or two, maybe three should be sufficient in most cases.

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?

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.

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.

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

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.

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.

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.

What is meant by the term "syntactic sugar"?

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

Characterize each statement below as true or false.

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

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

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

Problem 11: (18 points)

Answer any one, two or three of the following questions. If you choose to answer only one you'll need to have about three times as much depth or breadth as if you address all three.

Question 1:

Choose one of our three primary languages and take the position that it should be used to replace Java in CSc 127A/B. Present an argument in favor of this replacement. Your argument should point out aspects of your chosen language, and possibly the accompanying environment, that facilitate teaching fundamental concepts of programming and creation of interesting programming assignments. Also identify the greatest weakness of using the language in an introductory class and then address how to minimize the impact of that weakness.

Do not bother to address the fact that the replacement is not Java and therefore wouldn't take advantage of high school AP programs, be as widely accepted in industry, etc.

This question can be successfully answered using any of the three languages—you don't need to pick the one that you might think the instructor would pick.

Question 2:

A fundamental choice in a programming language is whether it does type checking when source code is compiled. Some languages forego analysis of types at compile time and instead only detect type mismatches when the code is executed. Programmers have a variety of opinions of the merit of the two approaches. Some favor compile-time checking for *all* software development; some avoid compile-time type checking like the plague. Describe your preferred position on the type-checking question and provide a rationale for it. Your position need not be polar—you might favor compile-time checking in some cases and not in others.

Question 3:

A programming language can be thought of as a system for describing computation. There are many examples of descriptive systems in the world but few if any descriptive realms have the variety of choices that are available in the realm of programming languages. What has motivated computer scientists to create so many different programming languages?

(Space for essay question responses.)

Problem 12: (6 points, EXTRA CREDIT)

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)` fails if S is anything other than a subscripting expression of one of the two forms above.

Assume the indexing values are integers, as shown in the examples above.

Assume you have a grammar rule `id(Ident)` that recognizes an identifier and instantiates `Ident` to atom, like `'s'` and `'line'` for the above.

Assume you have a grammar rule `digits(Digits)` that recognizes a sequence of digits and instantiates `Digits` to an atom consisting of the digits, just like in `listsum` on assignment 9.

Extra Credit Section (one half-point each unless otherwise indicated)

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

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

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

(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.

(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
```

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

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

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

(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)

(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?

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

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

(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.

(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. Hint: Here's a 0-point solution: `puts 100`