Last name

_____

CSC 372 Final Exam
Monday, May 9, 2016

READ THIS FIRST

Read this page now but do not turn this page until you are told to do so. Go ahead and fill in your last name in the box above.

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

The last five minutes of the exam is a "seatbelts required" period to avoid distractions for those who are still working. If you finish before the "seatbelts required" period starts, you may turn in your exam and leave. If not, you must stay quietly seated—no "packing up"— until time is up for all.

You are allowed no reference materials whatsoever.

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, like the name of a function or the order of function arguments, state the assumption and proceed.

Don't make problems hard by assuming that they need to do more than is specifically mentioned in the write-up or that the solution that comes to mind is "too easy."

If you're stuck on a problem, please ask for a hint. Try to avoid leaving a problem completely blank—that's a sure zero.

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 in the lower right hand corner of the top side of each sheet, checking to be sure your copy of the exam has all the pages.**

BE SURE to enter your last name on the sign-out log when turning in your completed exam.

I believe that at least one student will be taking a make-up exam, so I won't be posting solutions on Piazza. I'll instead park a copy in this directory: `http://cs.arizona.edu/~whm/more-pancakes-please/` Write it down! Note the ~ in `~whm`.

Problem 1: (6 points)

Cite three things about programming languages you learned by watching your classmates' video projects. Each of the three should be about a different language and have a bit of depth, as described in the Piazza post that announced this problem.

Problem 2: (1 point)

Write a Haskell function `doflip pancakes flip` that performs an `fsort`-style flip. Examples:

```
> doflip [3,5,4,2] 2
[5,3,4,2]

> doflip it 4
[2,4,3,5]
```

Assume the stack has at least one pancake and that `flip` is in range.

Problem 3: (3 points)

Write a **recursive** Haskell function `totlen list` that returns the total length of the strings in `list`.

```
> totlen ["just", "a", "test"]
9

> totlen []
0
```

Problem 4: (4 points)

Write a Haskell function `xout string` that replaces every letter (a-z) in `string` with an "x" of the same case. Non-letters are unchanged. Example:

```
> xout "Don't get stuck!  Keep moving!"
"Xxx'x xxx xxxxx!  Xxxx xxxxxx!"

> xout ""
""
```

Recall `isUpper` and `isLower` in `Data.Char`:

```
> isUpper 'A'
True

> map isLower "A t!"
[False,False,True,False]
```

You may assume `Data.Char` has been imported. It also has `isLetter`.

Problem 5: (2 points)

Write a folding function `f` such that the `foldr` call below behaves as shown, returning a list of the odd numbers in its last argument.

```
> foldr f [] [5,2,9,4,4,3,1]
[5,9,3,1]
```

Hint: The operation being performed is the same as `filter odd [5,2,9,4,4,3,1]` but you definitely don't want to use `filter` in your folding function!

Problem 6: (5 points)

Write a Ruby **iterator** `sbl(a, max)` ("strings by length") that first yields each one-character string in the array `a`. It then yields each two-character string in `a`. The process continues up through strings of length `max`. Strings are yielded in the order they appear in `a`. `sbl` always returns `nil`.

Assume that the array `a` contains only strings.

Restriction: You may not use `sort`. (If you did, you'd probably find that the sequence produced wouldn't be fully correct.)

The second example below demonstrates that a `max` of 2 limits results to one- and two-character strings.

```
>> sbl(["ab","b","a","aaa","test","bc","a"],10) { |s| p s }
"b"
"a"
"a"
"ab"
"bc"
"aaa"
"test"
=> nil

>> sbl(["ab","b","a","aaa","test","bc","a"],2) { |s| puts s.size }
1
1
1
2
2
=> nil
```

Remember: Don't use a `sort` method.

Problem 7: (4 points)

Write a Ruby method `pages_re` that returns a regular expression that matches a string iff the string consists of one or more comma-separated page specifications. A page specification is one of these three:

- A number, such as "`12`".
- Two numbers separated by a dash, such as "`1-3`".
- A number followed by a dash, such as "`2500-`".

A "number" is a sequence of one or more digits.

Examples:

```
>> pages_re =~ "1-10,15,20-,75"
=> 0

>> pages_re =~ "1,2,"
=> nil                (Has trailing comma)

>> pages_re =~ "5,ten"
=> nil                ("ten" is invalid)

>> pages_re =~ "Pages: 5,7-9"
=> nil                (Has "Pages: " at the beginning)
```

Problem 8: (11 points)

Write a Ruby program `mostfreq.rb` that reads lines on standard input and writes out each line followed by an annotation that shows which non-space character appears most frequently on the line, and how many times it appears. Input lines are right-padded with spaces to a length of 30. (Assume no line is longer than 30.)

```
% cal | ruby mostfreq.rb
      May 2016                ('6': 1)
Su Mo Tu We Th Fr Sa          ('S': 2)
 1  2  3  4  5  6  7          ('7': 1)
 8  9 10 11 12 13 14          ('1': 6)
15 16 17 18 19 20 21          ('1': 6)
22 23 24 25 26 27 28          ('2': 8)
29 30 31                      ('3': 2)

%
```

Important: Lines with no non-space characters, such as the last line of `cal` output above, have no annotation but are still padded to a length of 30.

In case of a tie, pick any one of the tying characters. (Note that the first line above has a six-way tie, for example.)

You may imagine that `Hash` has a `sort_by_value` method:

```
>> h = {"x" => 5, "y" => 10, "z" => 3}

>> h.sort_by_value
=> [["z", 3], ["x", 5], ["y", 10]]
```

Recall that `String#ljust` can be used to pad a string with blanks on the right:

```
>> "ab".ljust 5
=> "ab   "
```

Write your solution here, or on the next page.

(space for `mostfreq.rb` solution)

For reference:

```
% cal | ruby mostfreq.rb
      May 2016                ('6': 1)
Su Mo Tu We Th Fr Sa          ('S': 2)
 1  2  3  4  5  6  7          ('7': 1)
 8  9 10 11 12 13 14          ('1': 6)
...

%

>> {"x" => 5, "y" => 10, "z" => 3}.sort_by_value
=> [["z", 3], ["x", 5], ["y", 10]]
```

Problem 9: (8 points)

In this problem you are to implement a Ruby class named `Seq` that represents a sequence of values with a maximum length. The maximum length is specified when an instance is created.

Values are added to the end of the sequence using an overloaded << operator. If a value is added to a sequence that is already at maximum length, the first value in the sequence is discarded. The << operation returns `nil`.

`Seq#inspect` returns a string consisting of the values in the sequence separated by dashes. Vertical bars surround the full sequence.

Here's an example of interaction. Remember that `irb` uses `inspect` to display the result of each expression.

```
>> s = Seq.new 3
=> ||

>> s << 10
=> |10|

>> s << 20; s << 30
=> |10-20-30|

>> s << 40
=> |20-30-40|

>> s2 = Seq.new 10
=> ||

>> "this is a test of Seq".each_char {|c| s2 << c }
=> "this is a test of Seq"

>> s2
=> |e-s-t- -o-f- -S-e-q|
```

Implementation note: `Array#shift` discards the first element of an array.

Write your solution here, or on the next page.

(space for `Seq` solution)

```
>> s = Seq.new 3
=> ||

>> s << 10
=> |10|

>> s << 20; s << 30
=> |10-20-30|

>> s << 40
=> |20-30-40|

>> s2 = Seq.new 10
=> ||
```

Problem 10: (6 points)

Write the following simple Prolog predicates. There will be a half-point deduction for each occurrence of a singleton variable or failing to take full advantage of unification.

There are no restrictions.

(a) `fl_same(?L)` expresses the relationship that the first and last elements of `L` are identical. It should able to handle all possible combinations of instantiated and uninstantiated variables. `fl_same` fails if the list is empty. Examples:

```
?- fl_same([a,b,a]).
true.

?- fl_same(L).
L = [_G1191] ;
L = [_G1191, _G1191] ;
L = [_G1191, _G1194, _G1191] ;
L = [_G1191, _G1194, _G1197, _G1191] ;
...

?- fl_same([X,Y,5]).
X = 5.
```

(b) `revgen(+L,-X)` generates the elements of the list `L` in reverse order.

```
?- revgen([a,b,c,d], X).
X = d ;
X = c ;
X = b ;
X = a.
```

(c) Write the library predicate `member/2`. Examples:

```
?- member(X,[1,2]).
X = 1 ;
X = 2.

?- member(3,[1,2]).
false.
```

Problem 11: (6 points)

In this problem you are to write a Prolog predicate that is similar to the earlier Ruby problem "`sbl`" (strings by length).

`abl(+Atoms,+Max,-A)` first instantiates `A` to each one-character atom in `Atoms`, then each two-character atom in `Atoms`, etc. `abl` continues for atoms of lengths up through `Max`, if any are that long.

```
?- abl([abc,b,ab,zzz,a,zzzz,aa,c],10,A).
A = b ;
A = a ;
A = c ;
A = ab ;
A = aa ;
A = abc ;
A = zzz ;
A = zzzz ;
false.

?- abl([abc,b,ab,zzz,a,zzzz,aa,c],1,A).
A = b ;
A = a ;
A = c.
```

Similar to `sbl`'s restriction, you may not use any built-in sorting predicate, like `msort`.

Recall `atom_chars(+Atom, ?List_of_chars)`:

```
?- atom_chars(test,L).
L = [t, e, s, t].
```

Problem 12: (8 points)

Write a Prolog predicate `findrun(+L,+N,+X,-Pos)` that looks for `N`-long runs of `X` in `L`, instantiating `Pos` to the zero-based starting position of each run. Assume `N` is greater than zero.

```
?- findrun([a,b,b,c,d,b,b],2,b,Pos).
Pos = 1 ;
Pos = 5 ;
false.

?- findall(Pos,findrun([a,a,a,a,a,a],3,a,Pos),Positions).
Positions = [0, 1, 2, 3].

?- findall(Pos,findrun([a,b,a,a,b],1,a,Pos),Positions).
Positions = [0, 2, 3].
```

You may assume you have the `repl(+N,+Elem,-List)` predicate you wrote on assignment 9:

```
?- repl(a,5,L).
L = [a, a, a, a, a].
```

Please ask for a hint if you have trouble with this one!

Problem 13: (7 points)

Write a predicate `expand(+List,-Expanded)` that takes a list of (1) atoms and (2) structures of the form *`Atom*N`* and produces an "expanded" list. Assume the `N` terms are `>= 0`.

```
?- expand([a*2, b, test*3, none*0, here], L).
L = [a, a, b, test, test, test, here].

?- expand([],L).
L = [].
```

Problem 14: (11 points)

Write a Prolog predicate `reach(+Destination, +Jumps, +Fences, -Solution)` that finds a sequence of "jumps" on a Cartesian plane from `(0,0)` to the `Destination` (a `pos/2` structure), being sure that no jump lands on a fence.

`Fences` is a list of `fence` structures, like `[fence(x,3),fence(y,1)]`. The first element indicates there is a fence at `x = 3`. The second indicates there is a fence at `y = 1`. There may be any number of fences. Here's a representation of those two fences, and a destination of `pos(5,3)`, marked as `D`.

```
4 |           +
  |           +
3 |           +       D
  |           +
2 |           +
  |           +
1 |++++++++++++++++++++++
  |           +
  +-----------------------
      1   2   3   4   5
```

`Jumps` is a list of two-term `jump` structures that specify `x` and `y` distance, like `[jump(0,3),jump(1,1),jump(-3,2)]`. There may be any number of jumps.

The final jump must land exactly on the destination. Each jump can be used only once.

Here's a call of `reach` that produces two solutions:

```
?- reach(pos(5,3), [jump(1,1),jump(3,0),jump(1,2),jump(2,2)],
   [fence(x,3),fence(y,1)], Sol).   % Note: query is wrapped around
Sol = [jump(1, 2), jump(1, 1), jump(3, 0)] ;
Sol = [jump(1, 2), jump(3, 0), jump(1, 1)] ;
false.
```

Note that from `pos(0,0)`, doing `jump(1,1)` would land on the fence at `y = 1`.

Similarly, from `pos(0,0)`, doing `jump(2,2)` then `jump(1,1)` would land on the fence at `x = 3`.

Recall how `select/3` was used in the pit crossing example: `select(Plank, Planks, Remaining)`.

Note that the terms in a `jump` structure can be accessed by unifying it with a `jump` structure with uninstantiated variables:

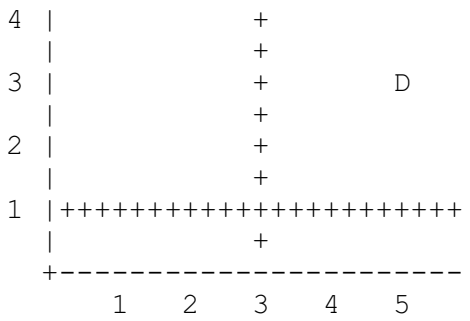
```
?- Jumps = [jump(3,4)], Jumps = [H|_], H = jump(X,Y).
Jumps = [jump(3, 4)],
H = jump(3, 4),
X = 3,
Y = 4.
```

`reach` produces all possible solutions in turn. (Just don't use any cuts and things should be fine!)

There's space for your solution on the next page.

(space for solution for `reach`)

For reference:



The final jump must land exactly on the destination. Each jump can be used only once.

```
?- reach(pos(5,3), [jump(1,1),jump(3,0),jump(1,2),jump(2,2)],
   [fence(x,3),fence(y,1)], Sol).   % Note: query is wrapped around
Sol = [jump(1, 2), jump(1, 1), jump(3, 0)] ;
Sol = [jump(1, 2), jump(3, 0), jump(1, 1)] ;
false.
```

Problem 15: (4 points) (one point each)

The following questions and problems are related to Prolog.

(1) When writing documentation for Prolog predicates, arguments are often prefixed with the symbols `+`, `-`, and `?`, such as `p(+X,?Y,-Z).` What does each of those three symbols mean?

(2) Write a sentence that expresses the relationship between the terms "fact", "clause", and "rule".

(3) Consider the following goal written by a novice Prolog programmer:

```
X is X + length(List)
```

What are **two** misunderstandings evidenced by that goal?

(4) What's the most important fundamental difference between a Prolog predicate like `append/3` and a function/method of the same name in Haskell, Ruby, or Java?

Problem 16: (7 points) (one point each unless otherwise indicated)

Below is a transcript of interaction with Standard ML of New Jersey, a functional programming language. Annotate the transcript with seven significant observations about Standard ML. Excellent or additional observations may earn up to a total of three points of extra credit. Here's an example of an insignificant observation: "There's a `map` function."

```
$ sml
Standard ML of New Jersey v110.69 [built: Mon Jun  8 23:24:21 2009]

- 5 + ~7;
val it = ~2 : int


- (1, 2.0, "three");
val it = (1,2.0,"three") : int * real * string


- explode "test";
val it = [#"t",#"e",#"s",#"t"] : char list


- implode it;
val it = "test" : string


- map (fn(n) => n * 3) [3, 1, 5, 9];  {- See above re map -}
val it = [9,3,15,27] : int list



- it::[];
val it = [(1,2.0,"three")] : (int * real * string) list



- fun f1 a b = [a,b];
val f1 = fn : 'a -> 'a -> 'a list



- fun f2 a b = [a = b];
val f2 = fn : ''a -> ''a -> bool list



- 3 + 4.5;
stdIn:1.1-1.6 Error: operator and operand don't agree [literal]
  operator domain: int * int
  operand:         int * real

- (hd [1,2,3], tl [1,2,3]);
val it = (1,[2,3]) : int * int list
```

Problem 17: (7 points) (one point each unless otherwise indicated)

Answer the following general questions.

(1) What's something significant related to programming languages that you remember from the JarWars video we viewed and discussed during the second-to-last class?

(2) Ralph Griswold said, "If you're going to invent a language, be sure to invent a language that you _____."

(3) What is the fundamental characteristic of a dynamically typed language?

(4) How does Icon avoid the "to versus through" problem that plagues string indexing in many languages and libraries?

(5) What's something significant about Icon you recall from the Icon by Observation exercise during the last class? (Hint: Don't cite your answer for the previous question!)

(6) With programming languages in general, what's the fundamental difference between a statement and an expression?

(7) Which of the three languages we covered this semester are you most glad we covered, and why?

Extra Credit Section (½ point each unless otherwise noted)

(1) In as few words as possible, what is "The Cathedral and the Bazaar"? (Mentioned in `a7` solutions.)

(2) In what month of the year 1891 were classes first held at The University of Arizona?

(3) With Prolog in mind, why is "camelcase variable" an oxymoron?

(4) The technology known as Leda was mentioned on Piazza. What is Leda?

(5) Draw an appropriate avatar for any one of the three languages we covered.

(6) Name a language that was once studied in depth in 372 but isn't any more.

(7) Who do you think whm will vote for in November's presidential election?

(8) Why did whm's solution for `pipes.pl` have the following line? `do(p) :- do(pipes).`

(9) (1 point) `mostfreq.rb` says to assume that `Hash` has a `sort_by_value` method. Write Ruby code that makes that be true.

(10) What did Knuth say about premature optimization?

(11) (1 point) Using only `append` and `length`, and no recursion, write a Prolog predicate `longer(A,B)`, which is true iff list `A` is longer than list `B`.

(12) (1 point) In SNOBOL4, how do you indicate where control should go if a statement fails?

(13) whm hates writing up Piazza posts with suggested readings! They usually don't align well with his slides and he wonders if anybody actually does any of the readings. Over the course of the full semester, how many hours do you estimate you spent doing the suggested readings? (This is just data collection, no right/wrong.)

(14) Write a joke about programming languages.

(15) Finish this sentence: "If I only remember one thing about 372 it will be ...".