Last name
____

CSC 372 Final Exam
Tuesday, May 12, 2015

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. I 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 arugments, 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 you have all nine sheets.**

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

Exam solutions will be posted here: (Write it down! Note the ~ in `~whm`)
`http://www.cs.arizona.edu/~whm/cons-me-up-942-cardboard-pancakes-to-flip/`

Quiz 18, May 12, 2015
2 questions; 4 points

1. For one point each, cite three things you learned from Dr. Proebsting's presentation in 372 on Tuesday, May 5, the last day of class.

2. On a scale of 1 to 5, with 1 being least important and 5 being most important, how would Dr. Proebsting rate the importance of language choice for a programming project?

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.

Problem 2: (3 points)

(1) (Two points) Imagine that a Haskell function `f` has the type `Char -> Bool -> Char`. Explain the meaning of that type in a way that demonstrates you have significant knowledge of Haskell.

Hint: here's an answer that's worth zero points: "`f` takes two arguments, a `Char` and a `Bool`, and returns a `Char`."

(2) (One point) Fully parenthesize the following Haskell expression.

```
f   1   2   g  +  x   x   +  f   g   f   1
```

Problem 3: (6 points)

Write a Haskell function `longpos :: [String] -> Int` that returns the <u>one</u>-based position of the longest string in a list of strings. Assume the list has at least one string. Don't worry about ties. Usage:

```
> longpos ["just","a","few","strings","here"]
4

> longpos ["hello!"]
1
```

You may use any Prelude functions you wish. You may also use `Data.List.sort`, with type `Ord a => [a] -> [a]`. Here's a reminder of how it works:

```
> sort [7,3,5]
[3,5,7]

> sort [(3,'a'),(1,'b'),(10,'c')]
[(1,'b'),(3,'a'),(10,'c')]
```

Hint: My solution is non-recursive. It uses `zip`.

Problem 4: (9 points)

Write a Ruby program that reads integers, fractions, and mixed numbers, one per line from standard input, and prints their sum when end of file is reached. Here's an example, with user input underlined and bold:

```
% ruby addmixed.rb
Number? 2
Number? 1/2
Number? 1 2/5
Number? ^D
Total: 3.9
```

Restriction: Your solution must be based on regular expressions; don't use `String#split` calls to break up the line.

Each input line must be solely an integer, a fraction, or a mixed number, with no leading or trailing spaces. Only one space may appear between the whole number and fraction in a mixed number. No negatives or decimal fractions are allowed. If a line doesn't satisfy any of those rules, "`Ignored: `*`<LINE>`*" is printed and the line is completely ignored. Examples:

```
% ruby addmixed.rb
Number? 3 7/100
Number? 1 2
Ignored: 1 2
Number? 10/20/30
Ignored: 10/20/30
Number? -50/3
Ignored: -50/3
Number? 1.5
Ignored: 1.5
Number?    1/3
Ignored: 1.5
Number? ^D
Total: 3.07
```

Implementation notes:

You might find it easier to use two or three matches than to have a single regular expression that accommodates integers, fractions, and mixed-numbers.

`"3 ".to_f` produces `3.0`.

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

(Space for solution for `addmixed.rb`)

Reminder of operation:

```
% ruby addmixed.rb
Number? 1 1/2
Number? 100
Number? 1/4
Number? 1 2/3/4
Ignored: 1 2/3/4
Number? ^D
Total: 101.75
```

Remember:

Solution must be regular expression-centric, not splits and length checks.
Each line must be only an integer, fraction, or mixed number; nothing more.

Problem 5: (9 points)

In this problem you are to write a Ruby version of `buy.pl` from assignment 9. The Ruby version reads a file named `buy.data` for information about items and discounts. The file has this format:

```
% cat buy.data
item|toaster|Deluxe Toast-a-matic|14.00
item|antfarm|Ant Farm|7.95
...
item|dip|French Onion Dip|1.29
discount|antfarm|20
discount|lips|40
discount|rshoes|10
...
```

Lines with information about items come first, with the fields separated by a vertical bar. Lines for discounts on items follow. The discount amount is the percentage to subtract from the regular price. With the specified 20% discount, the $7.95 Ant Farm will sell for $6.36 (`7.95 * 0.80`).

Use `Kernel#open` to open `buy.data` for reading. To read lines, use `File#gets`, which returns `nil` at end of file. Example:

```
>> f = open("buy.data")
=> #<File:buy.data>

>> f.gets
=> "item|toaster|Deluxe Toast-a-matic|14.00\r\n"
```

`buy.rb` is run with item identifiers as command line arguments. The output format is simple: each item's description is printed, followed by three dots, followed by the price (use `"%.2f"` as a `printf` format). A line with a total follows.

```
% ruby buy.rb antfarm catnip tiger twinkies twinkies
Ant Farm...6.36
50-pound bag of catnip...19.95
Sumatran tiger...749.95
Twinkies...0.75
Twinkies...0.75
Total: $777.76
```

If an unknown item is specified, a "price check" line is printed and execution is terminated:

```
% ruby buy.rb antfarm catfood catnip
Ant Farm...6.36
Price check for catfood!
%
```

Unlike the Prolog version, there are no "dontmix" specifications.

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

(space for `buy.rb`)

For reference:

```
% cat buy.data
item|toaster|Deluxe Toast-a-matic|14.00
...
item|dip|French Onion Dip|1.29
...

% ruby buy.rb lips lips dip
Chicken Lips...0.03
Chicken Lips...0.03
French Onion Dip...1.29
Total: $1.35
```

Problem 6: (6 points)

Implement assignment 9's `outin` as a Ruby iterator. Assume its argument is always a non-empty array.

```
>> outin([1,2,3,4,5]) {|x| puts x}
1
5
2
4
3
=> nil
```

For one point of extra credit, write `outin` so that it could be a mix-in for `Array`, and used like this:

```
>> "one two three".split.outin {|x| puts x}
one
three
two
=> nil
```

Problem 7: (5 points)

Ruby defines a meaning for `String * Fixnum` but the operation is not commutative—`Fixnum * String` produces an error:

```
>> 4 * "abc"
TypeError: String can't be coerced into Fixnum
```

For this problem you are to create a file named `symmul.rb` such that after `symmul.rb` is loaded, `Fixnum * String` works. Example:

```
>> load "symmul.rb"

>> 4 * "abc"
=> "abcabcabcabc"
```

Problem 8: (10 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.

Restriction: No library predicates may be used other than `is/2`. (But note exception for `init`!)

(a) `f(?X)` expresses the relationship that `X` is 3 or 4. Examples:

```
?- f(3).
true.

?- f(X).
X = 3 ;
X = 4.
```

(b) `head2(?L, ?Heads)` expresses the relationship that `Heads` is a list that consists of two copies of the head of `L`.

```
?- head2([1,2,3],X).
X = [1, 1].

?- head2([1,2,3],[1,2]).
false.

?- head2(L,[a,a]).
L = [a|_G2516].
```

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

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

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

(d) `sum(+L, -Sum)` instantiates `Sum` to be the sum of the elements in `L`, which are assumed to all be numbers.

```
?- sum([3,1,5],Sum).
Sum = 9.
```

(e) `init(?L, ?Init)` is an analog to Haskell's `init`: the list `Init` is all but the last element of `L`. `init` fails if `L` is empty. **Restriction EXCEPTION: Your solution for `init` may use append.**

```
?- init([a,b,c,d],I).
I = [a, b, c] .

?- init([a],I).
I = [] .
```

Problem 9: (4 points)

Write a Prolog predicate `zip(+L1, +L2, ?Pair)` that produces a series of `pair/2` structures:

```
?- zip([1,2,3,4],[a,b,c],R).
R = pair(1, a) ;
R = pair(2, b) ;
R = pair(3, c) ;
false.
```

The shorter list determines the number of results that are produced.

Restriction: You may not use any built-in predicates or write any helper predicates.

Problem 10: (6 points)

Write a Prolog predicate `swapends(?L1,?L2)` that expresses the relationship that `L1` is a copy of `L2` with the first and last elements swapped. `swapends` is only meaningful for lists with two or more elements.

```
?- swapends([a,b,c,d],L).
L = [d, b, c, a] ;
false.

?- swapends([1,2,3],L).
L = [3, 2, 1] ;
false.

?- numlist(1,7,Nums), swapends(Nums,R).
Nums = [1, 2, 3, 4, 5, 6, 7],
R = [7, 2, 3, 4, 5, 6, 1] ;
false.
```

Problem 11: (6 points)

On assignment 9 you wrote `outin(+L, -R)`, which worked like this:

```
?- outin([1,2,3,4,5],R).
R = 1 ;
R = 5 ;
R = 2 ;
R = 4 ;
R = 3 ;
false.

?- outin([1,2,3,4],R).
R = 1 ;
R = 4 ;
R = 2 ;
R = 3 ;
false.
```

On this problem you are write `inout(+L, -R)`, which generates elements in the opposite order from `outin`.

Examples:

```
?- inout([1,2,3,4,5],R).
R = 3 ;
R = 4 ;
R = 2 ;
R = 5 ;
R = 1.

?- inout([1,2,3,4],R).
R = 3 ;
R = 2 ;
R = 4 ;
R = 1.
```

IMPORTANT: You may use `outin` in your solution! This problem is intended to be easy; don't make it hard!

Problem 12: (12 points)

In this problem you are to write a Prolog predicate `cross/3` that finds a way to cross over a series of pits using wooden planks as bridges.

Here's an example that shows two pits:

```
----+      +--+  +-----
    |      |  |  |
    |      |  |  |
    +------+  +--+
    5     12  15 18
```

Pits are represented with `pit/2` facts, which have a starting position and a width:

```
pit(5,7)
pit(15,3)
```

There may be any number of `pit/2` facts. Pits never overlap. Pits always have ground between them.

Below is an example of a valid crossing of distance 20 that uses the sequence of planks `[3, 10, 10]`. Planks are shown with a vertical offset to make the lengths apparent.

```
===          ==========
   ==========
----+      +--+  +-------
    |      |  |  | ^
    |      |  |  | 20
    |      |  |  |
    +------+  +--+
    5     12  15 18
```

Planks must be placed so that both ends rest on solid ground, rather that having an end over a pit. Planks must extend continuously from a starting point to a specified length.

Here's an example of an invalid crossing, with the sequence `[9, 11]`. It's invalid because the two planks meet over a pit, at position 10. `[11,9]` and `[16,9]` would be invalid, too.

```
         ===========
=========
----+      +--+  +----
    |      |  |  | ^
    |      |  |  | 20
    |      |  |  |
    +------+  +--+
    5     12  15 18
```

A joint at position N between two planks is considered to be over a pit if start-of-pit <= N <= end-of-pit. Examples of invalid joint positions for the above pits are 5, 10, and 18. Valid joint positions include 4, 13, 14, and 19.

Continued on next page...

For reference, with two pits: `pit(5,7)` and `pit(15,3)`:

```
----+       +--+  +-----
    |       |  |  |
    |       |  |  |
    +-------+  +--+
    5      12  15 18
```

<u>The predicate you are to write is `cross(+Planks, +Distance, -Solution)`</u>. Examples of use:

```
?- cross([10,10,3],20,S).
S = [3, 10, 10] .

?- cross([9,11],20,S).
false.

?- cross([1,2,4,5,5,9],20,S).
S = [4, 9, 1, 5, 2] .
```

Assume that `Distance` is sufficient to cross all the pits. As the first example above demonstrates, the combined length of the planks may exceed `Distance`. It not necessary to use all the planks.

As the second example shows, `cross` fails if a crossing is not possible with the given planks.

Only the first result of `cross` is of interest—the user hit ENTER after each answer above.

<u>Remember that there may be any number of `pit/2` facts and the pits may be in any position.</u>

Hint: I use a helper, `layplanks(+Planks, +Current, +Distance, -Solution).`

Problem 13: (7 points) (one point each unless otherwise indicated)

<u>The following questions and problems are related to Prolog.</u>

(1) Haskell strings are actually just lists of characters. Similarly, Prolog lists are actually instances of a more basic element of Prolog. What's that element?

(2) What's an appropriate situation in which to use the "cut-fail" combination?

(3) Consider the following goal written by a novice Prolog programmer: `append(A,B,A)`. What's a likely problem with that goal?

(4) Consider this claim: Aside from what `is/2` provides, Prolog has nothing that corresponds to the concept of an expression like that found in Java, Haskell, and Ruby. Present an argument either in favor or against this claim. (2 points)

(5) What's the most important fundamental difference between a Prolog predicate like `member/2` and a function/method of the same name in Haskell, Ruby, or Java? (2 points)

Problem 14: (5 points) (one point each unless otherwise indicated)

The following is a Sather program, from `http://rosettacode.org/wiki/Sum_of_squares#Sather`

```
class MAIN is

  sqsum(s, e:FLT):FLT is
    return s + e*e;
  end;

  sum_of_squares(v :ARRAY{FLT}):FLT is
    return (#ARRAY{FLT}(|0.0|).append(v)).reduce(bind(sqsum(_,_)));
  end;

  main is
    v :ARRAY{FLT} := |3.0, 1.0, 4.0, 1.0, 5.0, 9.0|;
    #OUT + sum_of_squares(v) + "\n";
  end;

end;
```

Based on the code above, tell me five things about Sather. Excellent or additional observations may earn up to three points of extra credit.

Problem 15: (6 points)

Answer the following general questions.

(1) What is the fundamental characteristic of a statically typed language? (1 point)

(2) A language designer must decide whether an operation is to have a symbolic form, like `+`, `==`, and `x[y]` or an alphabetic form like `length`, `insert`, and `charAt`. What are some factors that must be considered when deciding between an symbolic or alphabetic form for an operation? (2 points)

(3) Imagine that for some reason all your knowledge and memories of two of the three languages we covered this semester must be erased! Which of the three languages do you wish to retain your memories of, and why? (3 points)

Extra Credit Section (½ point each unless otherwise noted)

(1) What programming language had a typo in the first example of the first book published about the languaage?

(2) The first machine named lectura was a VAX-11/785. Who picked the name "lectura"?

(3) What would be a better name for the Prolog 1000, and why?

(4) What does DWIM mean?

(5) What is the official name of Gould Simpon 942? (Ok to be funny!)

(6) Ralph Griswold is known for designing languages like SNOBOL4 and Icon but he's also known for his work related to the mathematical aspects of weaving. What's a <u>one word</u> connection between those two areas?

(7) In what year did Ralph Griswold found The U of A's Department of Computer Science?

(8) On Ralph's first day here he arrived to find a number of students waiting outside his office for advising. When he entered his office he found a common piece of office furniture was absent. What was it?

(9) There are many measures of success. What do you think must be true in order for a programming language to be considered a success?

(10) Under what circumstances should a programming language be considered dead?