Your last name	Last name of person sitting beside you
_____	_____

CSC 372 Final Exam
Tuesday, May 8, 2018

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. It will end at 5:30pm.

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/199341-more-videos-please/
```

Write it down! Note the ~ in `~whm`. If any trouble, DO NOT post on Piazza! Send me mail.

PLEASE, PLEASE, PLEASE RAISE YOUR HAND IF YOU'RE STUMPED ON A PROBLEM!

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: (6 points)

Given the following Haskell function definition,

```
f xs ys = foldr (++) [] (map m (zip xs ys))
    where
        m (a,b) = [a,b]
```

What does the following expression produce?

```
f ['h','e','l','l','o'] "hello"
```

a) `"hellohello"`
b) `["hh","ee","ll","oo"]`
c) `[('h','h'),('e','e'),('l','l'),('o','o')]`
d) `"hheellllloo"`
e) A type error of some sort.

Should your answer be wrong, showing your work or providing some explanation of your reasoning might earn some partial credit.

Problem 3: (4 points)

Write a Haskell function `again list`, of type `Eq t => [t] -> Bool`, that returns `True` if and only if the first element of `list` appears again elsewhere in the list.

`again` returns `False` if `list` has less than two elements.

RESTRICTION: You may **NOT** use the built-in function `elem` but you may write your own version of `elem`.

Examples:

```
> again [3,1,7,3,5]
True

> again [3,1,7]
False

> again [3]
False

> again []
False
```

Problem 4: (3 points)

Write a Ruby iterator `inorder(x,y,z)` that yields the values `x`, `y`, and `z` in ascending order. Assume that `x`, `y`, and `z` can be sorted with Ruby's `Enumerable#sort` method. `inorder` always returns `nil`.

```
>> inorder(10,2,4) { |x| print("x = #{x}\n") }
x = 2
x = 4
x = 10
=> nil

>> inorder(10,2,4) { puts "value" }
value
value
value
=> nil
```

Problem 5: (6 points)

This Ruby problem is similar to `btw.pl` from assignment 9. Write a Ruby iterator `btw(a,sep)` that yields copies of the array `a` with `sep` inserted between each element of `a` in turn. `btw(a,sep)` returns `a`.

Examples:

```
>> btw([1,2,3,4,5], "---") { |x| p x }
[1, "---", 2, 3, 4, 5]
[1, 2, "---", 3, 4, 5]
[1, 2, 3, "---", 4, 5]
[1, 2, 3, 4, "---", 5]
=> [1, 2, 3, 4, 5]

>> btw([10,20], "---") { |x| p x }
[10, "---", 20]
=> [10, 20]

>> btw([10], "---") { |x| p x }
=> [10]

>> btw([], "---") { |x| p x }
=> []
```

Problem 6: (5 points) 5

This problem is similar to `re.rb` on assignment 7. Write a method `zip_re` that returns a regular expression that matches the city/state/zip line of postal addresses, like these:

```
Tucson, AZ 85750
Beverly Hills, CA  90210
Walkertown, NC 27051
```

Use the following rules:

- City names start with a capital, contain any mix of letters and spaces, and end with a lower-case letter.
- A comma immediately follows the city; one or more spaces can follow the comma.
- The state is always two capital letters; one or more spaces can follow the state.
- The zip code is always five digits.
- Use anchors to ensure that no extraneous characters before or after the address are matched.

When the match is successful, the named groups `city`, `state`, and `zip` should contain those respective values.

Examples:

```
>> zip_re =~ "Tucson, AZ 85750"
=> 0

>> $~
=> #<MatchData "Tucson, AZ 85750" city:"Tucson" state:"AZ"
     zip:"85750">

>> zip_re =~ "Beverly Hills, CA  90210"
=> 0

>> $~
=> #<MatchData "Beverly Hills, CA  90210" city:"Beverly Hills"
     state:"CA" zip:"90210">

>> zip_re =~ "Walkertown NC 27051"     (has missing comma)
=> nil
```

Here's a reminder of how named groups work:

```
>> /(?<before>.+)-(?<after>.+)/ =~ "test-this" => 0

>> $~  => #<MatchData "test-this" before:"test" after:"this">
```

Problem 7: (9 points) 6

For this problem you are to write a Ruby program that implements a **very simple** simple line-oriented calculator. It supports three commands/operations:

variable=integer
: Assigns the integer value to the specified variable.

var1=var2+var3
: Assigns the sum of `var2` and `var3` to `var1`.

/vars
: Print all variables and their values. **Simplification**: Variables can be printed in any order.

Use these assumptions to **keep things simple**:

- Input lines are always well-formed. Calc does no error handling whatsoever.
- Input lines never contain any whitespace.
- Variables names are always a single lower-case letter.
- Variables are always assigned a value before being used.

An example of interaction follows. User input is in bold.

```
$ ruby calc.rb
calc> x=7
calc> y=-3
calc> z=x+y
calc> /vars
x: 7
y: -3
z: 4
calc> x=z+z
calc> /vars
x: 8
y: -3
z: 4
calc> ^D
```

Use `print "calc> "` to print the prompt.

Problem 8: (5 points) **7**

In this problem you are to implement a Ruby class named `Rope` that represents a length of rope that can be cut into two pieces.

An instance of `Rope` is made with an integer length:

```
>> r = Rope.new 7
```

A rope can be cut, but only once, and only if the length to be cut, an integer, is less than the length of the rope:

```
>> r.cut(10)          => false
>> r.cut(2)           => true
>> r.cut(1)           => false
```

The method `pieces` returns an array that represents the current state of the rope:

```
>> r.pieces           => [5, 2]
>> r2 = Rope.new(50)
>> r2.pieces          => [50]
>> r2.cut 1           => true
>> r2.pieces          => [49, 1]
```

Problem 9: (3 points) **8**

Write a Prolog predicate `ints_atoms(+L,-E)` that first instantiates `E` to each integer in `L`, and then instantiates `E` to each atom in `L`. Example:

```
?- ints_atoms([3,a,3+4,[1],b,7,q(1),z,10],E).
E = 3 ;
E = 7 ;
E = 10 ;
E = a ;
E = b ;
E = z ;
false.
```

Remember that `integer(+X)` succeeds iff `X` is an integer. Similary, `atom(+X)` succeeds iff `X` is an atom.

Problem 10: (5 points)

Write a predicate `middle(+List, -M)` that instantiates `M` to the middle element of the list `L`. `middle` fails if the list has an even number of elements.

RESTRICTIONS:

- Just like on `append.pl` on assignment 9, **the only predicates you can use are `length` and `append`**.
- `middle` cannot be recursive.

Remember that `\==`, `==`, `=`, `<`, `is`, `!`, and many more are predicates, too, and not allowed! Raise your hand if in doubt!

You can use the `[E1, E2, ..., En]` syntax for lists.

Examples:

```
?- middle([5,1,7,2,4],M).
M = 7 ;
false.

?- middle([1,2,3,4],M).
false.
```

Problem 11: (7 points)

9

For this problem you are to write a Prolog predicate `sumneq(+L, -Sum)`. The first argument, `L`, is a list of `vp/2` structures where both terms are integers. `sumneq` looks for `vp` structures with unequal terms, like `vp(3,4)` and `vp(-10,2)`, but not `vp(3,3)`. For each `vp` structure with unequal terms, `sumneq` computes the product of those two integer terms and then instantiates `Sum` to the sum of all those products.

My solution is one clause. It uses `findall`, `member`, `sumlist` and other predicates.

Examples:

```
?- sumneq([vp(3,2),vp(5,5),vp(3,1)],Sum).
Sum = 9.

?- sumneq([vp(1,1),vp(3,3),vp(-1,-2),vp(10,10)],Sum).
Sum = 2.

?- sumneq([vp(1,1)],Sum).
Sum = 0.

?- sumneq([],Sum).
Sum = 0.
```

Problem 12: (7 points) 10

Write a predicate `updown(+Ints, -Atom)` that instantiates `Atom` to a series of atoms in turn, based on the integers in `Ints`. Example:

```
?- updown([3,1,5], A).
A = 'up three' ;
A = 'down three' ;
A = 'up one' ;
A = 'down one' ;
A = 'up five' ;
A = 'down five'.
```

Specifically, for each integer in `Ints`, first an `'up...'` and then a `'down ...'` is generated.

Integers outside of the range 1-10 inclusive are silently excluded:

```
?- updown([7,11,372,5],A).
A = 'up seven' ;
A = 'down seven' ;
A = 'up five' ;
A = 'down five'.
```

Assume that ten `n/2` facts are present:

```
n(one,1).
n(two,2).
...
n(ten,10).
```

My solution uses `atom_concat/3`:

```
?- atom_concat(hello,world,A).
A = helloworld.
```

NOTE: Be sure that your solution generates all values, not just the first value.

Problem 13: (10 points) **11**

This is the problem like `connect` and pit-crossing that you were told to expect.

Write a Prolog predicate `seating(+Host, +Guests, -Seating)` that determines the seating for a dinner party to be attended by some number of conservatives and liberals.

`Guests` is a list of `con(`*`Name`*`,`*`Dist`*`)` and `lib(`*`Name`*`,`*`Dist`*`)` structures where *`Name`* is the person's name and *`Dist`* is how far from the center they are (always an integer greater than zero). `Host` is a `con/2` or `lib/2` structure as well. Examples:

```
lib(jane,1).     % liberal, distance 1 from the center
con(jim,3).      % conservative, 3 from the center
lib(tom,2).      % liberal, 2 from the center
```

Two people can be seated beside each other if (1) both are liberals, or (2) both are conservatives, or (3) the sum of their distances is four or less.

Given the three people above here's the seating situation:

`jane` and `tom` can sit beside each each other (both are liberals)
`jane` and `jim` can sit beside each other (differing beliefs, but total distance is <= 4)
`tom` and `jim` cannot sit beside each other (differing beliefs and total distance is > 4)

Two important points:

1. The table being used is round, so the first person seated, which is always the host, must able to sit beside the last person seated.
2. All guests must be seated. If there is no possible seating, `seating` fails.

Here's a simple case—a two-person party with a conservative host, John, and a liberal guest, Jane, but with a total distance that is less than or equal to four:

```
?- seating(con(john,1), [lib(jane,3)], S).
S = [con(john, 1), lib(jane, 3)].
```

In the following case, we can seat Jane between Mark and Bob:

```
?- seating(con(dave,4), [lib(jane,1), con(mark,2), con(bob,2)], S).
S = [con(dave, 4), con(mark, 2), lib(jane, 1), con(bob, 2)].
```

In some cases, seating is not possible. Here's a case where there's nobody that can sit beside Jane:

```
?- seating(con(dave,4), [lib(jane,1), con(mark,4), con(bob,4)], S).
false.
```

Here's a larger example:

```
?- seating(con(a,4), [lib(b,1), con(c,2), con(d,3), lib(e,2)], S).
S = [con(a, 4), con(c, 2), lib(e, 2), lib(b, 1), con(d, 3)].
```

There is space for your solution on the next page.

(Space for solution for `seating`)

For reference:

```
seating(+Host, +Guests, -Seating)

?- seating(con(dave,4), [lib(jane,1), con(mark,2), con(bob,2)], S).
S = [con(dave, 4), con(mark, 2), lib(jane, 1), con(bob, 2)].

?- seating(con(dave,4), [lib(jane,1)], S).
false.

?- seating(con(dave,3), [lib(jane,1)], S).
S = [con(dave, 3), lib(jane, 1)].
```

`select/3` is `select(?Element, ?List, ?Remaining)`

Remember:

- The host is seated first.
- Because the table is round, the last person seated must be able to sit beside the host.
- If all guests can't be seated, `seating` fails.

Problem 14: (10 points) (one point each unless otherwise indicated)

The following questions and problems are related to Prolog.

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

(2) Prolog goals were modeled using the "Four port model". What are the four ports?

(3) If a predicate with three clauses is being evaluated and the first clause fails, what does Prolog do?

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

(5) What did you find hardest to understand about Prolog?

(6) In your mind, what's the most interesting thing about Prolog?

(7) What does the predicate `trace/0` do?

(8) Write a Prolog clause that would generate a singleton warning.

(9) (Two points) What does the `findall` predicate do?

Problem 15: (2 points)

The following questions about SNOBOL4 are one point each. You may answer as many as you want but the maximum score on this problem is 2 points.

(1) Write a SNOBOL4 program that copies standard input to standard output, just like the `cat` command. (For a point of extra credit, implement `tac` instead, copying lines in reverse order—last line first, first line last. Hint: remember that `char(10)` produces a newline character.)

(2) Write a SNOBOL4 expression that concatenates the strings held in the three variables `x`, `y`, and `z`.

(3) In the following SNOBOL4 statement, which would start in column 1, what does the "`. x`" mean?

```
loop    line span(&digits) . x        :f(p)
```

(4) In the SNOBOL4 statement above, what does `:f(p)` mean?

(5) Tell me something else of significance about SNOBOL4.

Problem 16: (2 points)

The following questions about Icon are one point each. You may answer as many as you want but the maximum score on this problem is 2 points.

(1) Briefly describe Icon.

(2) What is the output of the following Icon expression? `write(2 > 3, 2+3, !"abc")`

(3) What was a fundamental design goal of Icon's string scanning mechanism?

(4) What's something that Prolog and Icon have in common?

(5) Tell me something else of significance about Icon.

Problem 17: (10 points) (one point each unless otherwise indicated) 15

Answer the following general questions.

(1) This is a "matching" problem. Draw a line connecting each of the following language facilities on the left with the best characterization of primitives vs. idioms/techniques for that language on the right.

Icon's string scanning	Many primitives but few idioms/techniques
Regular expressions in Ruby	Few primitives but many idioms/techniques
SNOBOL4 patterns	Few primitives and few techniques

(2) What is a fundamental characteristic of a statically typed language?

(3) What is the essence of the duck typing mindset?

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

(5) What is the basic idea of type inferencing?

(6) In any language you know, write an expression that has both a value and a side-effect. Cite the language, too.

(7) What is a fundamental benefit provided by a REPL?

(8) Tell me something about the concept of paradigms that is unrelated to programming paradigms in particular.

(9) (Two points) Which of the three main languages we covered this semester are you most glad we covered, and why?

Extra Credit Section (½ point each unless otherwise noted) **16**

(1) The slides had some quotes from prior 372 students. What's something you remember about one of them?

(2) What does the Prolog predicate `=..(?Struct, ?List)` do?

(3) What is the Chomsky hierarchy a hierarchy of?

(4) What's an example of a list that Prolog can sort but that Ruby and Python 3 can't?

(5) (One point) Present a brief argument either in favor of keeping Prolog as a 372 language or replacing it.

(6) Pick a nickname for one of the five languages we studied and explain why it's appropriate.

(7) Put the five languages we studied in order by their age, oldest first.

(8) What's a piece of wisdom from Ralph Griswold that `whm` passed along to you?

(9) What is a non-control, non-alphanumeric character and its ASCII code?

(10) Where in Gould-Simpson could leftover 372 handouts be found?

(11) The TAs were required to learn all of your names. What are the TAs' names? (First and last!)

(12) Late in the semester the class was invited to a sporting outing. What was the sport?

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

(14) (Zero points) `whm` booked his best of time of the semester ascending the steps to the eighth floor before Sunday's practice session! What was his time? (a) 25 seconds (b) 50 seconds (c) 1:40 (d) 3:20 (e) 6:40