CSc 372, Spring 1996
Mid-Term Examination
Tuesday, March 5, 1996

READ THIS FIRST

Do not turn this page until you are told to begin.

This examination consists of 20 problems presented on 12 numbered pages. When you are told to begin, first check to be sure you have all the pages.

On problems that ask you only to write code, you need not include any explanation in your answer if you are confident it is correct. However, if an answer is incorrect, any accompanying explanation may help you earn partial credit.

Please feel free to ask the instructor questions during the course of the exam. If you are unsure about the form or operation of a language construct that is central to a problem's solution, you are especially encouraged to ask the instructor about that construct.

If you're completely puzzled on a problem, the instructor may offer you a hint, at the cost of a deduction of points. If you agree to accept such a hint, the instructor will note that deduction on your paper.

Try to avoid leaving a problem completely blank—that will certainly earn no credit. If you can offer any sort of pertinent response it may earn you partial credit.

Problems marked with an asterisk (*) disproportionately hard with respect to their point values. You should do such problems last.

Except as otherwise noted, you may use any language elements you desire. For the Icon problems you may use `split`.

When you have completed the exam, give it to the instructor and enter your name on the exam sign-in log.

Problem 1 (6 points):

State a definition for the term "programming language".

Name a language element or capability one would almost certainly find in a language that supports imperative programming.

Name a language element or capability one would almost certainly find in a language that supports functional programming.

Problem 2 (4 points):

Write ML expressions having the following types:

```
int * string list
```

```
int * (int * (int * int) list)
```

Define ML functions named `f` and `g` having the following types. The functions need not perform any meaningful computation. You may define additional functions to help produce the desired types.

```
f: (int -> int) * int -> bool
```

```
g: int -> int -> int list
```

Problem 3 (6 points):

Consider the following ML function definitions:

```
fun h(x::xs) = x = 2
fun f(a,b,g) = h(g(a^"x")::b)
```

For each of the following identifiers, what type would ML infer, given the above definitions for `h` and `f`.

`f`?

`g`?

`h`?

`a`?

`b`?

`x`?

Problem 4 (3 points):

The following ML function definition is an example of using an exception.

```
exception NotOne;
fun f(n) = if n <> 1 then (raise NotOne) else n
```

Rewrite `f` to make better use of ML's pattern matching facilities.

Problem 5 (3 points):

What is meant by the ML warning "non-exhaustive match"?

What is one possible implication of the warning?

Write an ML function that would produce that warning.

Problem 6 (4 points):

Consider this fragment of a function definition:

```
fun f(x,y,z::zs) = ...
```

What values would be bound to `x`, `y`, `z`, and `zs` for this call, assuming that the body of the function `f` is compatible with the given values?

```
f([1]::[],(2,[3]),[[4,5,6]])
```

x?

y?

z?

zs?

Specify a function body for `f` that for the above argument tuple would produce the value 21. That is, instead of the "..." shown above, complete the function definition. (Hint: Don't make this too hard!)

What is the type of `f`, given the function body you specified in the previous part of this problem?

Problem 7 (5 points):

Write a function named `length` of type `int list -> int` that calculates the length of a list of integers. Examples of usage:

```
- length;
val it = fn : int list -> int

- length([1,2,1,2]);
val it = 4 : int
```

You may not use the built-in `length` function!

Problem 8 (8 points):

Write an ML function `avglen` of type `'a list list -> real` that computes the average number of elements in a list of lists. For example, if a list `L` contained an empty list and a list with five elements, `avglen(L)` would return 2.5. If the list is empty, raise the exception `EmptyList`.

You may wish to use the function `real`, of type `int -> real`, to convert an `int` into a `real`.

Examples of usage:

```
- avglen;
val it = fn : 'a list list -> real
- avglen([]);
uncaught exception EmptyList
- avglen([[]]);
val it = 0.0 : real
- avglen([[],[1]]);
val it = 0.5 : real
- avglen([[1],[1],[2,2]]);
val it = 1.33333333333333 : real
- avglen([[true,true],[true],[true,true,true]]);
val it = 2.0 : real
```

Problem 9 (8 points):

Write an ML function `F_L_eq` of type `''a list -> bool` that returns `true` if the first element in a list is equal to the last element, and returns `false` otherwise. If called with an empty list, `F_L_eq` should return `false`.

Examples of usage:

```
- F_L_eq;
val it = fn : ''a list -> bool
- F_L_eq([]);
val it = false : bool
- F_L_eq([1]);
val it = true : bool
- F_L_eq([1,2,1]);
val it = true : bool
- F_L_eq([1,2]);
val it = false : bool
- F_L_eq([[],[1],[2],[],[]]);
val it = true : bool
- F_L_eq(explode "abcde");
val it = false : bool
```

You may NOT use a function that reverses a list.

Problem 10 (4 points) (*)

Write an ML function `f(FL,VL)` that takes a list of functions (`FL`) and a list of values (`VL`) and produces a list of lists wherein the first list contains the results of applying each function in `FL` to the first elment in `VL`, and so forth, such that the Nth list contains the results of applying each function in `FL` to the Nth element in `VL`.

For example,

```
f([f1, f2, f3, ..., fM],[x1, x2, ..., xN])
```

would produce a value equivalent to an expression like this:

```
[[f1(x1), f2(x1), ..., fM(x1)],
 [f1(x2), f2(x2), ..., fM(x2)],
 ...
 [f1(xN), f2(xN), ..., fn(xN)]]
```

Problem 11 (3 points)

Lists in Icon and ML share a common syntax for literal specification of lists—the expression `[1,2,3]` specifies a simple list in both languages. But, ML places a constraint on lists that Icon does not—many lists that are valid in Icon are not valid in ML.

What is the constraint that ML places on lists that Icon does not?

Give an example of a list that is valid in Icon, but not valid in ML.

Problem 12 (2 points) (*)

Define ML functions named `f` and `g` having the following types. The functions need not perform any meaningful computation. You may define additional functions to help produce the desired types.

```
f: (string -> int) list -> int
```

```
g: string -> int -> real -> bool list
```

Problem 13 (4 points)

Icon's `reverse(s)` built-in function produces a reversed copy of a string `s`. Write an Icon procedure `Reverse(s)` to do the same thing. Of course, `Reverse` may not use `reverse`.

Problem 14 (8 points)

Write an Icon procedure `Point(s)` that takes a string representation of a point in 2D cartesian space such as `"(10,20)"` and if the string is well-formed, returns the X and Y coordinates as integers in a list. If the string is not well-formed, `Point(s)` fails.

`Point` is not required to handle negative coordinates, but if it can, that is fine. That is, for the call `Point("(-10,-20)")`, it is acceptable either if `Point` fails or if it returns the list `[-10,20]`.

Examples:

```
][ Point("(1,2)");
   r := L1:[1,2]  (list)
][ Point("(100,200)");
   r := L1:[100,200]  (list)
][ Point("10,20");
Failure
][
```

Problem 15 (8 points)

Write an Icon program that reads standard input and produces a histogram of line lengths encountered. For example, for the file

```
abc
def
ghij
klm
nop
qr
st
uv
wxyz
```

The following output would be produced:

```
Length  Occurrences
2       ***
3       ****
4       **
```

If you choose to use a table in your solution, note that if `x` is a table, `sort(x)` produces a list of the form `[[k1,v1],[k2,v2],...,[kN,vN]]` where each `ki/vi` represents a key/value pair in the table. The pairs are ordered by increasing value of `ki`.

Problem 16 (9 points):

Imagine a file with a format such as this:

```
02.sting
01.just te
10. this out
```

Each line begins with a sequence number that is followed by a dot. An arbitrary string follows the dot. Note that there is no whitespace at the beginning of the line.

In a given file all sequence numbers are the same length, but the sequence number length may vary from file to file—do not assume a length of two for sequence numbers. Sequence numbers are always specified with leading zeros.

Write a program that reads such a file on standard input, assembles the lines in order based on the sequence numbers, and writes to standard output a sequence of fixed length lines. The full set of sequence numbers may be non-consecutive, as shown above, but there will be no duplicated sequence numbers.

The length of the output lines is determined by a command line argument, which if not specified defaults to 10.

Assuming that the program is called `assemble`, here are some invocations on an input file containing the above three lines:

```
% assemble <assemble.in
just testi
ng this ou
t
% assemble 15 <assemble.in
just testing th
is out
% assemble 1000 <assemble.in
just testing this out
%
```

Another input file and sample invocations:

```
% cat assemble.in2
0004.xy
0001.abcdefg
0002.hijklmnop
0005.z...
0003.qrstuvw
% assemble 25 < assemble.in2
abcdefghijklmnopqrstuvwxy
z...
```

Problem 16 (space for solution):

Problem 17 (3 points):

Write an Icon procedure `cons(x,y)` that approximates the ML operation `x::y` as closely as possible. If the approximation is poor, explain the difficulty.

Problem 18 (3 points) (*):

Write a procedure `size(x)` that has the same result as `*x` for values of `x` that are a `string`, `list`, or `table`. You may not use the `*` operator in your solution.

Examples:

```
][ size("");
   r := 0  (integer)
][ size("abc");
   r := 3  (integer)
][ size([1,2,3,4]);
   r := 4  (integer)
][ size(table(""));
   r := 0  (integer)
```

Problem 19 (6 points) (*):

Write an Icon program to read standard input and print out the largest integer found in the input. Consider an integer to be simply a series of consecutive digits; don't worry about issues with negative numbers. The program's output should be simply the largest integer. If no integers are found the program should output "`No integers`".

```
% bigint
On February 14, 1912, Arizona
became the 48th state.
^D
1912
% bigint
A test with no integers
in the input.
^D
No integers
% bigint
-100-1000-100000
^D
100000
```

Problem 20 (3 points) (*):

Icon has language elements to support imperative programming, but could Icon adequately support functional programming? Present an argument in support of your answer.

CSc 372, Spring 1996
Final Examination
Tuesday, May 7, 1996

READ THIS FIRST

Do not turn this page until you are told to begin.

This examination consists of 14 problems and an extra credit section presented on 15 numbered pages. When you are told to begin, first check to be sure you have all the pages.

On problems that ask you only to write code, you need not include any explanation in your answer if you are confident it is correct. However, if an answer is incorrect, any accompanying explanation may help you earn partial credit.

Please feel free to ask the instructor questions during the course of the exam. If you are unsure about the form or operation of a language construct that is central to a problem's solution, you are especially encouraged to ask the instructor about that construct.

If you're completely puzzled on a problem, please ask for a hint.

Try to avoid leaving a problem completely blank—that will certainly earn no credit. If you can offer any sort of pertinent response, it may earn you partial credit.

Except as otherwise noted, you may use any language elements you desire.

Please do not write on the back of any page—ask for additional sheets of paper if you run out of room.

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

Course grades are due at 5pm on Thursday, May 9. Shortly after that, if not before, you will be notified of your final grade via e-mail. Graded exams with solutions will be available in the Computer Science main office by Monday, May 13.

Print your name below and when told to begin, put your initials in the lower right hand corner of each page, being sure to check that you have all the pages.

Name: _____

Problem 1 (10 points):

What is the difference between a class and an object?

What is the proper way to view the relationship between function members and data members in a C++ class?

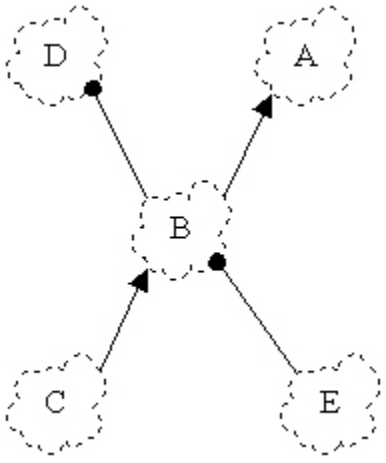
Under what circumstances should a data member in a C++ class be public?

What is the difference between the class relationships of inheritance and containment?

As described by the instructor, what is the primary benefit of inheritance?

Problem 2 (5 points):

Write a set of C++ class declarations that capture the relationships shown in this Booch Notation class diagram:



Here's a start—a complete declaration for `A`:

```
class A { };
```

Problem 3 (20 points):

In this problem you are to fully implement a C++ class named `ReplStr`. The abstraction represented by `ReplStr` is a character string that consists of a "base" string repeated (replicated) a specific number of times. For example, the string "`ababab`" might be represented with `ReplStr("ab", 3)` or `ReplStr("abababab", 1)`.

`ReplStr` should have this interface:

```
class ReplStr {
    public:
     ReplStr();
          //
          // Creates a string of length zero

     ReplStr(String s, int n);
          //
          // Creates a string consisting of n
          // concatenations of s.

     int Length();
          //
          // Produces the length of the replicated String.

     String GetString();
          //
          // Produces the replicated string as an
          // instance of the String class.
     };
```

The binary operations of equality (==), inequality (!=), and concatenation (+) can be applied to pairs of `ReplStr`s.

Examples of use:

```
ReplStr r1("abc", 2);
int l1 = r1.Length();       // should be 6
String s1 = r1.GetString(); // should be "abcabc"

ReplStr ab1("ab",3), ab2("abababab",1);
ReplStr x6("x",6);

int t1 = ab1 == ab2;    // Should be 1
int t2 = x6 == ab1;     // Should be 0

int t3 = ab1 != ab2;    // Should be 0
int t4 = x6 != ab1;     // Should be 1

ReplStr s2("abc",2), s3("xy",3), s4;
```

```
s4 = s2 + s3;   // s4 should represent "abcabcxyxyxy"
                // s2 and s3 should be unchanged
```

ReplStrs can be inserted in ostreams. The lines:

```
cout << "ab1 = '" << ab1 << "', ab2 = '" << ab2
     << "', x6 = '" << x6 << "'" << endl;

cout << "s4 = '" << s4 << "'" << endl;
```

should produce the output:

```
ab1 = 'ababab', ab2 = 'ababab', x6 = 'xxxxxx'
s4 = 'abcabcxyxyxy'
```

Tasks in this problem:

(1) Define a set of private data members for ReplStr.

(2) Define implementations for the two constructors and the Length and GetString methods.

(3) Define implementations for the overloaded ==, !=, +, and << operators. You may add additional public and/or private methods to ReplStr.

(4) If needed, define implementations for the assignment operator and the copy constructor. If either or both are not needed, explain why not.

(5) Describe a pattern of usage for which your implementation would have poor performance characteristics and explain the difficulty. If you believe your implementation has uniformly good performance characteristics, make an argument to support that claim.

Note that ReplStr utilizes the String class studied during the course. In addition to the String methods studied in class and implemented in assignment 6, you may assume that an insertor for String exists and that the operations of == and != are defined for pairs of Strings.

Problem 3—space for solution

Problem 4 (5 points):

Fully implement classes `Bicycle` and `Tricycle` as follows.

A `Bicycle` or `Tricycle` can only be created given an owner's name represented by a `char *`. Examples:

```
Bicycle b1("Jimmy"), b2("Susan");
Tricycle t3("Mary"), t4("Bobby"), t5("Joey");
```

A `Bicycle` or `Tricycle` can be asked for the name of its owner or the number of wheels it has:

```
char* o1 = b1.GetOwner(); // Should produce "Jimmy"
int w1 = b1.GetNumWheels(); // w1 should be 2
int w2 = t3.GetNumWheels();  // w2 should be 3
```

Write a function `CountWheels` that can be used to count the number of wheels in a collection of bicycles or tricycles. Two possible uses of it:

```
Bicycle *bikes[] = { &b1, &b2, 0 };
Tricycle *trikes[] = ( &t1, &t2, &t3, 0 };
int c1 = CountWheels(bikes); // c1 should be 4
int c2 = CountWheels(trikes); // c2 should be 9
```

Your implementation of `CountWheels` should be able to accommodate possible future classes such as `Unicycle` or `TandemBicycle`, assuming that they also have a `GetNumWheels` method.

If you wish, you may introduce an additional class or classes to simplify your solution.

Note that you must specify all the code necessary to compile and run the examples given.

Problem 5 (5 points):

Write a Prolog predicate `flip(L1,L2)` that if given a list such as `[a,b,c,d,e,f]` as `L1` it will instantiate `L2` to be the list `[b,a,d,c,f,e]`. That is, it flips the position of each element in a list on a pair-wise basis. `flip` should fail if given a list with an odd length.

Examples of usage (blank lines have been elided):

```
| ?- flip([1,2,3,4], L).
L = [2,1,4,3] ? ;
no
| ?- flip([1,2,3], L).
no
| ?- flip([a,b], L).
L = [b,a] ? ;
no
| ?- flip([],L).
L = [] ? ;
no
```

Problem 6 (5 points):

State in your own words the relationship expressed by each of the rules of the `append` predicate:

```
append([],L,L).
append([X|L1],L2,[X|L3]) :- append(L1,L2,L3).
```

For reference, here are some examples of usage:

```
| ?- append([1,2],[3,4],L).
L = [1,2,3,4] ? ;
no
| ?- append([1,2],L,[1,2,3,4]).
L = [3,4] ? ;
no
```

Problem 7 (5 points):

Write a Prolog predicate `trim(L, SL, NL)` that describes the relation that the list `NL` is the list `L` with the list `SL` removed if `SL` is a prefix or suffix of `L`. Note that if `SL` is a both a prefix and a suffix, two alternatives should be generated.

Examples of usage (blank lines have been elided):

```
| ?- trim([a,b,c,d],[a,b],L).
L = [c,d] ? ;
no
| ?- trim([a,b,c,d],[b,c,d],L).
L = [a] ? ;
no
| ?- trim([a,b,c,d,a],[a],L).
L = [b,c,d,a] ? ;
L = [a,b,c,d] ? ;
no
| ?- trim([a,b,c],[a,b,c],L).
L = [] ? ;
L = [] ? ;
no
| ?- trim([a,b,c],[],L).
L = [a,b,c] ? ;
L = [a,b,c] ? ;
no
| ?- trim([a,a,b,c,a,a],[a],L).
L = [a,b,c,a,a] ? ;
L = [a,a,b,c,a] ? ;
no
| ?- trim([a,a,b,c,a,a],[a,a],L).
L = [b,c,a,a] ? ;
L = [a,a,b,c] ? ;
no
```

Problem 8 (5 points):

Do EITHER part (8a) or part (8b), but not both. If you work on both, CLEARLY mark which solution you wish to have graded.

(8a) Write a predicate `maxint/2` that if given a list of integers will find the largest element in the list. `maxint` should fail if given an empty list.

Examples of usage:

```
| ?- maxint([5,10,15],M).
M = 15 ? ;
no
| ?- maxint([10,5,15],M).
M = 15 ? ;
no
| ?- maxint([10],M).
M = 10 ? ;
no
| ?- maxint([],M).
no
```

(8b) Write a predicate `sum_check/2` that determines if a given integer is the sum of a list of integers.

Examples of usage:

```
| ?- sum_check(10, [1,2,3,4]).
yes
| ?- sum_check(10, [1,2,3,4,5]).
no
| ?- sum_check(0, []).
yes
```

Hint: Note that a query such as '`4 is 2+2`' is valid and succeeds.

Problem 9 (5 points):

Write a Prolog predicate `find_missing/3` that can be called with any two arguments instantiated and if the two supplied arguments are equal, the uninstantiated argument is instantiated to the value of the other two.

Examples of usage:

```
| ?- find_missing(X,1,1).
X = 1 ? ;
no
| ?- find_missing(1,2,X).
no
| ?- find_missing(3,X,3).
X = 3 ? ;
no
| ?- find_missing([a],X,[a|[]]).
X = [a] ? ;
no
```

Problem 10 (5 points):

Consider two predicates that express relationships about employees in a factory. The first is `knows/2`:

```
knows(A,B)
```

This expresses the relationship that `A` knows `B`. For example, if Bob knows Mary that might be expressed with this fact:

```
knows(bob,mary).
```

The second predicate is `shift/2`:

```
shift(bob,day).
shift(mary,night).
```

The above two facts indicate that Bob works on the day shift and Mary works on the night shift.

Using the two predicates `knows` and `shift`, do the following:

(1) Write a Prolog query that expresses *Is there is a person who knows both Bob and Mary?*:

(2) Write a Prolog query that expresses *Does Mary know anyone who works on a different shift than she?*:

(3) Define a new clause for `knows/2` that expresses *A person knows everyone who works on the same shift.*

(4) Define a predicate `by_shift` that prints a list of employees by shift. Usage:

```
| ?- by_shift.
Day:
   bob
   mary
   joe
Night:
   bill
   jim
   sam
   susan
```

At your discretion, `by_shift` may either succeed or fail.

Problem 11 (3 points):

Write an ML function `allsame(L)` of type `''a list -> bool` that returns true if all the elements in a list are equal and false otherwise. `allsame([])` should return false.

Examples of usage:

```
- allsame;
val it = fn : ''a list -> bool

- allsame([1]);
val it = true : bool

- allsame([1,2,3]);
val it = false : bool

- allsame([1,1,1,1]);
val it = true : bool

- allsame([1,1,2]);
val it = false : bool

- allsame([]);
val it = false : bool
```

Problem 12 (2 points):

Consider this function definition:

```
fun f(x,g) = x::g(x-1)
```

What types would ML infer for:

f?

g?

x?

Problem 13 (5 points):

Write an Icon program `tac` to read a text file on standard input and print on standard output the lines of the file in reverse order—last line first; first line last.

For the input file:

```
just
a
test
right here
```

The output should be:

```
right here
test
a
just
```

Write your solution to the right of the above example.

Problem 14 (20 points):

Write a letter to a friend that discusses each of the four languages studied in this class. You may assume that your friend has had exposure to languages like C and Pascal. You may also assume that your friend has knowledge of data structures such as linked lists and trees.

For each language be sure to:

(1) Describe in broad terms the type of problems that the language is well-suited for.

(2) Describe at least three distinctive elements of the language and give an example of the use of each. Choose elements that are not closely related to each other—for example, don't simply cite three different data types, or three different control structures.

(3) Show a situation where the language presents a clear advantage over doing the same task in C.

And, mention which of the languages is your favorite, and why.

Problem 14—additional space for answer

Extra Credit Problems

(1 point) Andrew Koenig's paper, *An Anecdote About ML Type Inference*, describes an incident in which ML's type inferencing system produced a suprising result. What was that result?

(2 points) Name five programming languages that originated before 1980.

(2 points) In C, write a recursive version of the library function `strlen(char*)`. You may not use any library functions, or perform any assignment, augmented assignment, or increment/decrement operations. In other words, write `strlen` using a functional style.

(1 point) Who was the person that first described the notion of partially evaluated functions, such as that provided with currying in ML?

(1 point) Name a popular operating system in which Prolog is utilized.

(5 points) Do a great job on question 14.

CSc 372, Fall 1996
Mid-Term Examination
Monday, October 21, 1996

READ THIS FIRST

Do not turn this page until you are told to begin.

This examination consists of nine problems and an extra credit section presented on twelve numbered pages.

On problems that ask you only to write code, you need not include any explanation in your 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 are unsure about the form or operation of a language construct that is central to a problem's solution, you are strongly encouraged to ask the instructor about it.

If you're completely puzzled on a problem, please ask for a hint.

Try to avoid leaving a problem completely blank—that will certainly earn no credit.

On the C++ problems you may use any language constructs you desire.

On the ML problems you may use only language constructs covered in class. **You may not use the `hd` or `tl` functions. The #N operator to extract elements from a tuple (e.g. #2(t)) may not be used.**

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

Print your name below and when told to begin, put your initials in the lower right hand corner of each page, being sure to check that you have all the pages.

This is a fifty minute exam with a total of 100 points. You should therefore average two points of completed problems per minute in order to finish in the allotted time.

Name: _____

For the following ML-related problems you may assume you have at your disposal the `m_to_n`, `map`, `reduce`, and `filter` functions as discussed in class. You may also use any of the built-in functions discussed in class such as `size`, `length`, `map`, etc.
If there is a function you would like to use but you are in doubt as to its suitability, please ask.

You are encouraged to use helper functions (either in `let`s or not) in your solutions.

As mentioned on the cover page, `hd`, `tl`, and the `#n` operator are off-limits.

Problem 1: (2 points each; 4 points total)

State the type of each of the two following expressions, or if the expression is not valid, state why.

```
(1,2,(1=2,1+2),"x")
```

```
[[],[1],[1,2]]
```

Problem 2: (2 points each; 8 points total)

Consider this ML function definition:

```
fun f(a,b,c) = (b, a(b), c(a)) = ("x", 2, [1])
```

State the type that will be deduced for each of the following: (2 points each)

`a`:

`b`:

`c`:

The type of the result produced by `f`:

Problem 3: (4 points)

Write a function `f` that has the type `int -> int -> int -> int -> bool`. You may use literals (constants) but you may not use any explicit type specifications such as `x:int`.

Problem 4: (12 points)

Write a function `ints_to_strs(L,c)` that for the `int list L` and the `string c` produces a list of strings where the *i*th element in the resulting list is a string composed of *N* replications of `c` where *N* is the *i*th element in the input list.

Examples:

```
- ints_to_strs;
val it = fn : int list * string -> string list

- ints_to_strs([2,1,3], "x");
val it = ["xx", "x", "xxx"] : string list

- ints_to_strs([2,1,3], "ab");
val it = ["abab", "ab", "ababab"] : string list
- ints_to_strs(m_to_n(1,5), "a");
val it = ["a", "aa", "aaa", "aaaa", "aaaaa"] : string list
- ints_to_strs([1,2], "");
val it = ["", ""] : string list
- ints_to_strs([], "x");
val it = [] : string list
```

You may assume that all the integers in the given list are greater than zero.

Problem 5: (12 points)

Write a function `len(L)` of type `string list list -> int` that produces the total number of characters in all the strings in the lists contained in `L`. (12 points)

Examples:

```
- len;
val it = fn : string list list -> int

- len([["a","b"], ["xx", "yyy"]]);
val it = 7 : int

- len([["1"], ["2"], ["3"], ["4","5","6"]]);
val it = 6 : int

- len([]);
val it = 0 : int

- len([[]]);
val it = 0 : int
```

Problem 6: (10 points)

Write a function `pair(L)` that accepts an `'a list` as an argument and produces a list of tuples containing consecutive pairs of elements from the list `L`. That is, the first tuple in the result list should contain the first and second elements from the list. The second tuple should contain the third and fourth elements from the list, etc.

If the list is of an odd length, e.g., three elements, the exception `OddLength` should be raised.

Examples:

```
- pair;
val it = fn : 'a list -> ('a * 'a) list

- pair([1,2,3,4,5,6]);
val it = [(1, 2), (3, 4), (5, 6)] : (int * int) list

- pair(["a","A","two","too","up","down"]);
val it = [("a", "A"), ("two", "too"), ("up", "down")]
    : (string * string) list

- pair([1,2,3,4,5,6,7]);
Uncaught exception: OddLength

- pair([]);
val it = [] : ('a * 'a) list
```

Problem 7: (26 points)

In this problem you are to implement a class named `Eater`. Instances of `Eater` are initialized with an integer that is the `Eater's` capacity expressed in food units. This creates an `Eater` with a capacity of ten food units:

```
Eater e1(10);
```

An `Eater` can be told to eat some number of food units:

```
e1.Eat(7);
e1.Eat(5);
```

If an `Eater` consumes more than its capacity, it burps. In the above case, the second invocation of `Eat` would produce this output as a side effect:

```
burp!
```

Each burp reduces the volume currently retained in the `Eater` by the capacity of the `Eater`. After the burp, `e1` would then be holding two units.

Continuing the example, a large feeding may produce several burps:

```
e1.Eat(29);
```

Output:

```
burp!
burp!
burp!
```

Note that three burps are produced because the `Eater` had two units remaining after the first burp.

An `Eater` may be inserted into an output stream:

```
cout << e1 << endl;
```

Output: (note that `0x5eb10` is the memory address of the `Eater`)

```
Eater at 0x5eb10 has burped 4 times
```

Another small example:

```
Eater a(1); a.Eat(3);
```

Output:

```
burp!
burp!
burp!
```

A longer example:

```
Eater E(10);
```

```
E.Eat(5);
cout << "..A.." << endl;
E.Eat(12);
cout << "..B.." << endl;
E.Eat(2);
cout << "..C.." << endl;
E.Eat(12);
cout << "..D.." << endl;
cout << E << endl;
```

Output:

```
..A..
burp!
..B..
..C..
burp!
burp!
..D..
Eater at 0x5eaf8 has burped 3 times
```

You may assume the specified capacity of an `Eater` will be greater than zero. You may assume that a given amount to eat will be non-negative.

Problem 8: (10 points)

Create an abstract base class named `Food` and create two derived classes named `Burger` and `Fries`. Any instance of a subclass of `Food` can be queried for the

number of food units it is:

```
Burger b;
Fries f;

cout << "A burger has " << b.GetUnits() << " units" << endl;
cout << "An order of fries has " << f.GetUnits() << " units"
     << endl;
```

Output:

```
A burger has 25 units
An order of fries has 15 units
```

Note: Every `Burger` has 25 units and every `Fries` has 15 units.

Extend `Eater` so that an `Eater` can be told to eat an instance of any subclass of `Food`.

```
Burger b;
Fries f;
Eater e2(20);

cout << "..1.." << endl;
e2.Eat(&b);
cout << "..2.." << endl;
e2.Eat(&f);
cout << "..3.." << endl;
```

Output:

```
..1..
burp!
..2..
burp!
..3..
```

Operation summary for `Eater`, `Food`, `Burger`, and `Fries`—be sure that your implementations support all these operations:

```
Eater e(10);
e.Eat(100);
cout << e << endl;
Burger b;
Fries f;
e.Eat(&b);
e.Eat(&f);
int x = b.GetUnits(), y = f.GetUnits();
```

[Space for solution for problem 8]

Problem 9: (2 points each; 14 points total)

Answer each of the following questions related to object-oriented programming in general and C++ in particular.

What is the difference between a class and an object?

What is the purpose of a constructor?

What is the purpose of a destructor?

If a class has no member functions, how many data members should it have?

Challenge or defend this statement: In C++, one possible reason to have a public data member is to avoid the overhead of calling a function to access that data.

Challenge or defend this statement: The purpose of member functions is to provide well-controlled access to data members.

Describe a benefit provided by using inheritance.

Optional Extra Credit Problems:

What is a practical reason to prefer pattern matching rather than using the `hd` and `tl` functions in SML? (2 points)

Write the minimum amount of code necessary to compile and run this code: (4 points)

```
X x1, xa[10], *xp;
X x2 = x1;
x1 = xa[0];
cout << &*xp << endl;
```

Could C++ be used effectively for functional programming? Present an argument to support your answer—don't just say "yes" or "no". (5 points)

Write in C a version of `int strcmp(char *, char *)` that uses no assignment operators of any form, nor the `++` or `--` operators. Recall that `strcmp` returns 0 if the strings are equal and a non-zero value if the strings are not equal. (3 points)

[Additional work space]

CSc 372, Fall 1996
Final Examination
Monday, December 16, 1996

READ THIS FIRST

Do not turn this page until you are told to begin.

This examination consists of 14 problems and an extra credit section presented on 14 numbered pages.

On problems that ask you only to write code, you need not include any explanation in your 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 are unsure about the form or operation of a language construct that is central to a problem's solution, you are strongly encouraged to ask the instructor about it.

If you're completely puzzled on a problem, please ask for a hint.

Try to avoid leaving a problem completely blank—that will certainly earn no credit.

Unless otherwise noted on a given problem you may use whatever language elements you desire.

Print your name below and when told to begin, put your initials in the lower right hand corner of each page, being sure to check that you have all the pages.

This is a two hour exam with a total of 100 regular points and 28 points of extra credit problems. You should therefore average slightly better than one point of completed problems per minute (1.067 p/m) in order to finish all problems in the allotted time.

Name: _____

Problem 1 (12 points):

For each of ML, C++, Icon, and Prolog, cite three elements in the language that are unique to that language among this group of four. Elements should be of a broad nature rather than focused on narrow details such as reserved words, built-in functions, and the like. Think in terms of the elements in the language that you would first mention if describing it to someone. Remember: Elements must be unique to a particular language. Note: Read the next problem before you answer this one.

Problem 2 (8 points):

Select one element from EACH set of three in the previous problem and describe a benefit it provides to the programmer.

In the following Prolog problems you may assume that you have at your disposal all of the predicates that we covered in class, such as `len/2 (length)`, `member/2`, `append/3`, `last/2`, etc.

Problem 3 (5 points):

Write a predicate `permute(L,P)` that for a list `L` of 1, 2, or 3 elements instantiates P to each permutation of the elements of L. A one element list `[a]` has one permutation: `[a]`. A two element list `[1,2]`, has two permutations: `[1,2]` and `[2,1]`. A three element list `[a,b,c]` has six permutations: `[a,b,c]`, `[a,c,b]`, `[b,a,c]`, `[b,c,a]`, `[c,a,b]`, and `[c,b,a]`. You may generate the permutations in any order. Hint: You can do this with a predicate with nine clauses, none of which are rules. You may abbreviate `permute` as `p` and use ditto marks (") where useful.

Examples:

```
?- permute([1,2],P).
P = [1,2] ? ;
P = [2,1] ? ;
no

?- permute([1,2,3],P).
P = [1,2,3] ? ;
P = [1,3,2] ? ;
P = [2,1,3] ? ;
P = [2,3,1] ? ;
P = [3,1,2] ? ;
P = [3,2,1] ? ;
no
```

Problem 4 (3 points):

The instructor's implementation of `roman/2` used a series of facts of this form:

```
rdval('I',1).
rdval('V',5).
rdval('X',10).
etc.
```

Consider an attempt at an ML function to provide the same functionality as `rdval/2`:

```
fun rdval("I") = 1
 |  rdval("V") = 5
 |  rdval("X") = 10
etc.
```

Without particular regard to usage in `roman/2`, is the ML function a good approximation of the `rdval/2` predicate? Be sure to justify your answer.

Problem 5 (7 points):

Write a predicate `sum_ints(L,Sum)` that produces a sum of the integers in list `L`. `L` might contain things other than integers, but they should be ignored. Examples:

```
?- sum_ints([1,3,5], Sum).
Sum = 9 ? ;
no

?- sum_ints([a,1,b,2,c,3], Sum).
Sum = 6 ? ;
no

?- sum_ints([], Sum).
Sum = 0 ? ;
no

?- sum_ints([a,b,c], Sum).
Sum = 0 ? ;
no
```

Note that the predicate `integer/1` can be used to see if a term is an integer:

```
?- integer(5).
yes

?- integer(a).
```

```
no

?- integer([1,2]).
no
```

Hint: Be sure your solution accommodates being asked for alternatives (As shown, none should be produced.)

Problem 6 (6 points):

Write a predicate `assemble(L,Segments)` that describes the relationship that the list `L` can be assembled from two of the lists in `Segments`. Examples:

```
?- assemble([1,2,3,4], [[1,2,3],[4],[3,4],[1,2]]).
yes

?- assemble([1,2,3,4], [[1,2,3],[3,4],[1,2]]).
yes

?- assemble([a,b], [[a],[b],[a,b],[a,b,c],[]]).
yes

?- assemble([a,b,c], [[a,b],[b,c],[c,a],[b]]).
no
```

Problem 7 (10 points):

Imagine that at your disposal is a predicate `make_change(CoinStock, Amount, Coins, NewStock)` that is used to calculate the number of nickels, dimes, and quarters necessary to add up to a given amount. Examples:

```
?- make_change([5,5,10,10,25,25], 30, C, N).
C = [5,25]
N = [5,10,10,25] ?

?- make_change([5,10,10,25], 5, C, N).
C = [5]
N = [10,10,25] ?

?- make_change([10,10,25], 18, C, N).
no
```

`make_change` fails if exact change can't be produced. `make_change` will produce alternatives and is not guaranteed to produce the best result first:

```
?- make_change([5,5,5,5,5,25], 25, C, N).
C = [5,5,5,5,5]
N = [25] ? ;

C = [25]
N = [5,5,5,5,5] ? ;
no
```

In this problem you are to write a predicate `cfa/2` (change for amounts) with this form: `cfa(Amounts, Coins, CoinLists)`. `Amounts` is a list of amounts for which change is to be produced from the list `Coins`. `cfa` instantiates `CoinLists` to a list of lists where each element is a list of coins totaling the amount in the corresponding position in `Amounts`. Examples:

```
?- cfa([30,5,20], [5,5,10,10,25,25], CoinLists).
CoinLists = [[5,25], [5], [10,10]] ?

?- cfa([30,5,5], [5,5,10,10,25,25], CoinLists).
no

?- cfa([10,10,10,10], [5,5,5,5,5,5,10], CoinLists).
CoinLists = [[5,5], [5,5], [5,5], [10]] ?
```

Note that the second case fails because the nickels ran out.

Important: The task at hand is to write `cfa` using `make_change`. `cfa` should be able to accommodate a list of amounts of any length. **If you can't work out a solution that handles any number of amounts you may write a solution that handles only lists of three amounts for a score of 6 points rather than the full 10 points for this problem. If you can't do that either, you can implement all of `member(X,L)`, `length(L,Len)`, `last(L,Last)`, and `append(L1,L2,L3)` for a score of four points.**

For reference, here again are some of the examples from the previous page:

```
?- make_change([5,5,10,10,25,25], 30, C, N).
C = [5,25]
N = [5,10,10,25] ?

?- make_change([5,10,10,25], 5, C, N).
C = [5]
N = [10,10,25] ?

?- make_change([10,10,25], 18, C, N).
no

?- cfa([30,5,20], [5,5,10,10,25,25], CoinLists).
CoinLists = [[5,25], [5], [10,10]] ?

?- cfa([30,5,5], [5,5,10,10,25,25], CoinLists).
no

?- cfa([10,10,10,10], [5,5,5,5,5,5,10], CoinLists).
CoinLists = [[5,5], [5,5], [5,5], [10]] ?
```

Don't forget: Your task is to write `cfa`; you may assume that you have `make_change` at your disposal.

In the following Icon problems you may assume that you have at your disposal the full set of built-in functions and the `split` procedure.

Problem 8 (8 points):

Write an Icon procedure `extract(s, m, n)` that extracts a portion of a string that represents a hierarchical data structure. `m` is a major index and `n` is a minor index. Major sections of the string are delimited by slashes and are composed of minor sections separated by colons. Here is a sample string:

`/a:b/apple:orange/10:2:4/xyz/`

It has four major sections which in turn have two, two, three and one minor sections.

A call such as `extract(s, 3, 2)` should locate the third major section (`"10:2:4"` in the string above) and return the second minor section therein (`"2"`). If either section number is out of bounds, `extract` should fail. `m` and `n` may be assumed to be integers greater than zero. `s` may be assumed to be well-formed.

Examples (with `ie`):

```
][ s := "/a:b/apple:orange/10:2:4/xyz/";

][ extract(s, 1, 1);
   r := "a"  (string)

][ extract(s, 1, 2);
   r := "b"  (string)

][ extract(s, 3, 3);
   r := "4"  (string)

][ extract(s, 4, 1);
   r := "xyz"  (string)

][ extract(s, 4, 2);
Failure

][ extract(s, 5, 1);
Failure
```

Problem 9 (6 points)

Write an Icon program that reads, on standard input, a list of words, one per line, and prints the words that contain the letters a, b, and c in that order. The letters need not be consecutive and there may be more than one occurrence of each. The only requirement is that in order for a word to be printed it must contain an "a" followed by a "b" that is followed by a "c". You may assume that the input is strictly lower-case.

Examples of words satisfying the desired condition:

```
acrobatic

elasmobranch

swashbuckler
```

Problem 10 (6 points)

Write an Icon procedure `revby2(s)` that reverses the string `s` on a character pair-wise basis and returns the resulting string. `revby2` should fail if `s` has an odd number of characters. **NOTE: Your solution must use string scanning. You may not use string subscripting (`s[i]`) or sectioning (`s[i:j]`), or the `*` operator.**

Examples (with `ie`):

```
][ revby2("12345678");
   r := "78563412"  (string)

][ revby2("abcdefghijklmnopqrstuvwxyz");
   r := "yzwxuvstqropmnklijghefcdab"  (string)

][ revby2("");
   r := ""  (string)

][ revby2("123");
Failure
```

Problem 11 (7 points):

Write an Icon program that prints on standard output, one per line, each minute of the day in the form `12:22pm`. The program should produce 1440 lines of output (24 times 60). The desired output, with key portions and shown and other portions elided, is as follows. Note that to conserve space the output is shown here in three columns but your solution should produce output in a single column.

```
12:00am          10:02am          1:02pm
12:01am          10:03am           ...
  ...              ...            9:58pm
12:58am          11:58am          9:59pm
12:59am          11:59am         10:00pm
 1:00am          12:00pm         10:01pm
 1:01am          12:01pm         10:02pm
 1:02am          12:02pm           ...
  ...              ...           11:58pm
 9:58am          12:58pm         11:59pm
 9:59am          12:59pm
10:00am           1:00pm
10:01am           1:01pm
```

Note that the first time printed is `12:00am` and the last is `11:59pm`. You may find the `right(s, width, pad_character)` function handy:

```
][ right(3, 2, "0");
   r := "03"  (string)

][ right(3, 2, " ");
   r := " 3"  (string)
```

Recall that in an Icon expression with multiple generators, generators are resumed in LIFO order: the generator that most recently produced a result is the first one resumed to produce a new result.

Problem 12 (4 points):

Write an Icon program `rev` that reads a file redirected to standard input and writes out the lines in the file in reverse order—last line first, first line last.

Example:

```
% cat in.dat
line 1
number two
the third line
% rev < in.dat
the third line
number two
line 1
```

Problem 13 (9 points):

Write an ML function `samesums(L)` of type `(int * int * int) list -> bool` that tests whether all the 3-tuples in `L` have the same sum.

```
- samesums;
val it = fn : (int * int * int) list -> bool

- samesums([(1,1,1),(3,0,0),(5,3,~5)]);
val it = true : bool

- samesums([(1,1,1),(3,0,1)]);
val it = false : bool

- samesums([(1,1,1)]);
val it = true : bool

- samesums([]);
val it = true : bool
```

Problem 14 (9 points)

Write code for a C++ class named `X` that exhibits the following elements:

A private default constructor.

A public constructor that takes two `int`s and stores their sum in a private data member of type `int`.

A public member function `int f()` that produces the sum computed by the `(int, int)` constructor or zero if this object was created by the default constructor.

Write an insertor for `X` that simply inserts (for example) "`an X at 0x7FFFEBBE`" where the address is the location of the instance of `X` in memory.

Write code to create five instances of X as follows: one local variable, one dynamically allocated instance, and a local variable that is an array of three instances. Show this code in a complete function.

Write code to invoke `X::f` for each of the five instances created in the above step and print the sum of the five return values.

EXTRA CREDIT SECTION

EC 1 (5 points):

Name five programming languages that originated before 1985.

EC 2 (5 points max):

For one point each, name a programming language and the person generally credited as being the designer of the language.

EC 3 (1 point):

What is the instructor's favorite programming language?

EC 4 (1 point):

Name a popular operating system in which Prolog plays a role in system configuration.

EC 5 (2 points)

Languages can be grouped according to various aspects of the language. For example, ML, Icon, and Prolog all have automatic memory management and C++ does not. Group the languages we studied according to their type checking philosophy.

EC 6 (3 points)

Write an Icon program that reads a list of words like that described in problem 12 and prints out the words that consist of solely of the hex digits a-f. Examples of such words: added, beef, dead, facade.

EC 7 (1 point)

Among ML, C++, Icon, and Prolog, which is your favorite?

EC 8 (2 points)

Of all that we covered, which one language feature did you find most interesting?

EC 9 (5 points):

In the same style as problems 1 and 2, name three differences between Java and C++ and for any one of those differences explain the benefit provided to the programmer.

EC 10 (2 points):

Why are static class members an essential element of Java?

EC 11 (1 point):

What is the Prolog 1000?

CSc 372, Spring 1997
Mid-term Examination
Thursday, March 13, 1997

READ THIS FIRST

Do not turn this page until you are told to begin.

This examination consists of 12 problems presented on 16 numbered pages.

On problems that ask you only to write code, you need not include any explanation in your 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 are unsure about the form or operation of a language construct that is central to a problem's solution, you are strongly encouraged to ask the instructor about it.

If you're completely puzzled on a problem, please ask for a hint.

Try to avoid leaving a problem completely blank—that will certainly earn no credit.

On the C++ problems you may use any language constructs you desire.

On the ML problems you may use only language constructs covered in class. **You may not use the `hd` or `tl` functions. The #N operator to extract elements from a tuple (e.g. #2(t)) may not be used.**

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

Print your name below and when told to begin, put your initials in the lower right hand corner of each page, being sure to check that you have all the pages.

This is a seventy-five minute exam with a total of 100 points.

Name: _____

For the following ML-related problems you may assume you have at your disposal all the functions discussed in class, such as `m_to_n`, `reduce`, and `filter`. You may also use any of the built-in functions discussed in class such as `size`, `length`, `map`, etc.

If there is a function you would like to use but you are in doubt as to its suitability, please ask.

You are encouraged to use helper functions (either in `let`s or not) in your solutions.

As mentioned on the cover page, `hd`, `tl`, and the `#n` operator are off-limits.

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

State the ***type*** of each of the following expressions, or if the expression is not valid, state why. For example, the type of the expression `3+4` is `int`.

```
([1, 2], [3.0, 4.0])
```

```
(map size) (explode "abcd")
```

```
("x", (2, "y"), ([3, "z"]))
```

```
(length, [size, length o explode])
```

Problem 2: (2 points)

Consider this ML function definition:

```
fun f(x) = [x 1, 2]
```

What type will be deduced for `f`?

Problem 3: (1 point each; 5 points total)

Consider these two ML function definitions:

```
fun f(_) = 1
fun g(a,b,c) =  if a = f(c) then b else [c,a]
```

State the type that will be deduced for each of the following:

`a`:

`b`:

`c`:

The return type of `f`:

The return type of `g`:

Problem 4: (3 points)

Write a function `f` that has the type `int -> int list -> bool`. You may use literals (constants) but you may not use any explicit type specifications such as `x:int`. The behavior of the function is not important—the task is simply to produce the desired type.

Problem 5: (9 points)

Write a function `tossbig(L,N)` of type `string list * int -> string list` that produces a list consisting of the elements in `L` that have a size less than `N`.

Examples:

```
- tossbig;
val it = fn : string list * int -> string list

- val SL = ["just", "a", "test", "right", "now"];
val SL = ["just","a","test","right","now"] : string list

- tossbig(SL, 4);
val it = ["a","now"] : string list

- tossbig(SL, 5);
val it = ["just","a","test","now"] : string list

- tossbig(explode "abc", 2);
val it = ["a","b","c"] : string list

- tossbig(explode "abc", 1);
val it = [] : string list
```

Problem 6: (9 points)

Write a function `samesums` of type `(int * int) list -> bool` that produces true if adding the two elements of each tuple in turn produce the same sum.

Examples:

```
- samesums;
val it = fn : (int * int) list -> bool

- samesums([(0,3)]);
val it = true : bool

- samesums([(0,3),(2,1),(4,~1)]);
val it = true : bool

- samesums([(1,2),(3,0),(4,0),(5,0),(1,2)]);
val it = false : bool

- samesums([(0,0),(1,~1),(2,~2),(3,~3)]);
val it = true : bool
```

Problem 7: (12 points)

Write a function `block(w,h)` of type `int * int -> unit` that prints a block of `x`'s having a width of `w` and height of `h`. You may assume that `w` and `h` are greater than or equal to 1.

Your solution must be NON-RECURSIVE in nature. You may use recursive routines such as `m_to_n` and `map`, but you may not invent any recursive routines yourself.

Examples:

```
- block;
val it = fn : int * int -> unit

- block(5,3);
xxxxx
xxxxx
xxxxx
val it = () : unit

- block(2,2);
xx
xx
val it = () : unit

- block(1,1);
x
val it = () : unit
```

Problem 8: (6 points)

Write a function `c_block` of type `int -> int -> unit` that behaves like `block`, but that allows partial application. You may use `block` in your solution.

Examples:

```
- c_block;
val it = fn : int -> int -> unit

- c_block 5;
val it = fn : int -> unit

- it 3;
xxxxx
xxxxx
xxxxx
val it = () : unit

- c_block 2 2;
xx
xx
val it = () : unit
```

Problem 9: (30 points)

In this problem you are to implement a C++ class that models a television set that has a list of favorite channels, each with a preferred volume. The television set has channels ranging from 2 to 83 and volume levels of 1, 2, 3, ..., 10.

```
class TV {
  public:
    TV();
       //
       // Initializes a TV, setting the channel to 2 and
       // the volume to 5

    void SetVol(int vol);
        //
        // Sets the volume to the level vol.  Attempts to set
        // the volume to less than 0 or more than 10 are ignored.

    void SetChan(int chan);
        //
        // Sets the current channel to chan.  Attempts to set the
        // channel to be less than 2 or more than 83 are ignored.

    void SetFav();
        //
        // Marks the current channel as a favorite channel and
        // records the current volume as the preferred volume
        // for the channel.

    void SetFav(int vol);
        //
        // Marks the current channel as a favorite channel and
        // establishes vol as the preferred volume for the
        // channel.  If the volume is less than 0 or more than
        // 10, SetFav is ignored.  SetFav does not change the
        // the current volume.

        // For both forms of SetFav, if the current channel has
        // already been selected as a favorite, the only effect
        // is to possibly change the preferred volume.  Note that
        // there is no way to indicate that a channel is no
        // longer a favorite.

    void NextFav();
        //
        // Goes to the next higher channel marked as a favorite
        // and sets the preferred volume.  After channel 83,
        // NextFav wraps around to 2 and continues, if necessary.
        //
        // If no favorites are marked, no change is made.

    void Status();
        //
        // Prints current channel and volume.
  };
```

An annotated example of operation follows. The only output produced by the program are the boldface lines beginning "`Channel is ...`".

```
     void main()
     {
          TV tv;

          tv.Status();
Channel is 2, volume is 5
 (Initial conditions)

          tv.SetChan(9);
          tv.SetFav(10);

          tv.Status();
Channel is 9, volume is 5
 (Channel 9 was established as a favorite with preferred volume of 5)

          tv.SetChan(12);
          tv.NextFav();
          tv.Status();
Channel is 9, volume is 10
 (Starting from channel 12, just marked as a favorite, the search for
  the next favorite channel goes through 83 and back around to channel
  9, with a preferred volume of 10.)

          tv.SetChan(13);
          tv.SetVol(3);
          tv.SetFav();
          tv.Status();
Channel is 13, volume is 3

 (At this point, channels 9 and 13 have been marked as favorites, with
     volumes of 10 and 3, respectively.)

          tv.NextFav();
          tv.Status();
Channel is 9, volume is 10

          tv.NextFav();
          tv.Status();
Channel is 13, volume is 3

          tv.SetChan(8);
          tv.SetFav(0);
          tv.SetVol(4);
          tv.SetChan(50);

          tv.NextFav();    tv.Status();
Channel is 8, volume is 0
          tv.NextFav();    tv.Status();
Channel is 9, volume is 10
          tv.NextFav();    tv.Status();
Channel is 13, volume is 3
          tv.NextFav();    tv.Status();
Channel is 8, volume is 0
     }
```

Your task in this problem is to specify the private portion of the `TV` class and implementations for each method. You need not recopy the public portion shown above. Hints: (1) For a central data structure my solution uses an 84 element `int` array with the Nth element holding a preferred volume if channel N is a favorite channel. (2) The purpose of this problem is to determine if you can put together a C++ class. With that purpose in mind, I don't intend to deduct for minor programming problems such as off-by-one errors.

[Additional space for solution for problem 9]

Problem 10: (5 points)

With our `String` class in mind, overload the `%` operator so that an expression such as `s % c` produces an `int` that is the zero-based position of the first occurrence of the character `c` in the `String` `s`. If `s` contains no occurrences of `c`, negative one should be produced. You may choose to implement the operator as a member function or a free-standing function. Recall that the C library function `char *strchr(const char *s, char c)` returns the *address* of the first occurrence of `c` in `s`.

Your solution should simply consist of the function or method definition itself.

Example:

```
String str = "testing";
int p1 = str % 's';
int p2 = str % 'x';

cout << "p1 = " << p1 << ", p2 = " << p2 << endl;
```

Output:

```
p1 = 2, p2 = -1
```

Problem 11: (2 points each; 6 points total)

Briefly answer each of the following questions related to object-oriented programming in general and C++ in particular.

(a) What is the difference between a class and an object?

(b) As you know, the destructor for a class `X` is a method named `~X`. The rationale for this choice of name is that "destruction is the complement of construction". However, the instructor has contended on several occasions that the relationship between constructors and destructors is in fact somewhat asymmetrical. Describe that asymmetry.

(c) What is the key benefit provided by in-line functions in C++?

Problem 12: (9 points)

In this problem you are to implement a class called `Ring` that is a subclass of the `Shape` class described in the slides. A ring can be thought of as a disk with a circular center region removed. In the following diagram, the black region is a `Ring`:

An instance of `Ring` is created by specifying the radius of the inner circle and the radius of the outer circle. A ring having the proportions of the one shown above might be created in this way:

```
Ring r(0.75, 1.0);
```

The radius of the inner circle is specified first and may be assumed to be non-negative and not greater than the radius of the outer circle. The area of a ring is the difference of the areas of the two circles. The perimeter of a ring is the perimeter of the outer circle.

Your first task in this problem is to specify a full class definition and member implementations for `Ring` as a subclass of `Shape`. You are to work with this definition of `Shape`:

```
class Shape {
  public:
    virtual double Area() = 0;
    virtual double Perimeter() = 0;
    virtual void Print(ostream& o) = 0;
    };
```

If your implementation of `Ring` requires changes in `Shape`, show those changes.

Here is an example of `Ring` in operation:

```
void main()
{
    Ring r(.75, 1.0);
    r.Print(cout);

    Ring r1(0.0, 1.0);
    Ring r2(0.0, 2.0);
    Ring r3(1.0, 2.0);

    r1.Print(cout);
    r2.Print(cout);
    r3.Print(cout);
}
```

Output:

```
Ring; area = 1.37445, perim = 6.28319
Ring; area = 3.14159, perim = 6.28319
Ring; area = 12.5664, perim = 12.5664
Ring; area = 9.42478, perim = 12.5664
```

Recall the `SumOfAreas` and `Biggest` functions presented in the slides, which follow below. Your second task in this problem is to indicate what changes that are required to these routines in order for them to accommodate `Rings` in addition to `Circles` and `Rectangles`, which the routines already handle.

```
double SumOfAreas(Shape *shapes[])
{
        double area = 0.0;

        for (int i = 0; shapes[i] != 0; i++) {
                Shape *sp = shapes[i];
                area += sp->Area();
                }

        return area;
}

Shape* Biggest(Shape *shapes[])
{
     Shape *bigp = shapes[0];

     for (int i = 0; shapes[i] != 0; i++) {
          Shape *sp = shapes[i];
          if (sp->Area() > bigp->Area())
               bigp = shapes[i];
          }
     return bigp;
}
```

If you wish, you may use the `Circle` class in your implementation of `Ring`.

[Space for solution for problem 12]

CSc 372, Spring 1997
Final Examination
Friday, May 16, 1997

READ THIS FIRST

Do not turn this page until you are told to begin.

This examination consists of 11 problems and an extra credit section presented on 18 numbered pages.

On problems that ask you only to write code, you need not include any explanation in your 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 are unsure about the form or operation of a language construct that is central to a problem's solution, you are strongly encouraged to ask the instructor about it.

If you're completely puzzled on a problem, please ask for a hint.

Try to avoid leaving a problem completely blank—that will certainly earn no credit.

On the C++, Icon, and Prolog problems you may use any language constructs you desire. On the ML problems you may use only language constructs covered in class.

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

Print your name below and when told to begin, put your initials in the lower right hand corner of each page, being sure to check that you have all the pages.

This is a two-hour exam with a total of 100 regular points and 10 points of extra credit questions.

Name: _____

Problem 1: (2 points each; 20 points total)

Based on the material covered in class, in what language is the expression `[10,20|30]` valid? What does it mean?

What is the type of the following ML expression?

```
[([length],[1,2,3])]
```

In ML, implement the well-studied `map` function in a way that yields the same type as the built-in version of map. This is the type desired:

```
('a -> 'b) -> 'a list -> 'b list
```

In your own words, what does the term "object-oriented programming" mean?

Cite a fundamental benefit of inheritance in C++.

What's unusual about this Prolog predicate?

```
f(L) :- g(L), fail, !.
```

Describe in English the form of the list `L` that would satisfy this predicate:

```
p(L) :- append(L1,L1,L).
```

Consider this version of member which has been instrumented with calls to `write/1`:

```
member(X,L) :- write('A'), L = [X|_].
member(X,[_|T]) :- write('B'), member(X,T), write('C').
```

What would be printed in response to the following query?

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

Write the `append/3` predicate.

Name two significant things that Prolog has in common with any one of the other languages that we studied.

Problem 2 (31 points):

Write an Icon program `ckconn` that analyzes test run output of `connect` from assignment seven and determines for each test case if the output shown is a suitable configuration of cables.

The data to be processed looks like this

```
case 1: data is '[[[m,10,f],[f,7,m]],m,15,f]'
F----------MF-------M

case 2: data is '[[[m,10,f],[f,7,m]],m,15,m]'
Cannot connect

case 3: data is '[[[m,5,m],[f,20,f],[m,10,m]],f,10,f]'
M-----M

case 4: data is '[[[m,5,m],[f,20,f],[m,10,m]],f,10,f]'
M-----MM----------M

case 5: data is '[[[m,5,f]],f,10,f]'
M----------M
```

The output of `ckconn` is the input data augmented with an error indication if the cable configuration shown is invalid in some way. For example, the case 3 result is invalid because the cable isn't long enough. The case 4 result is invalid because of the male/male connection in the middle. The case 5 result is invalid because there is no ten-foot M/M cable to be used. If an invalid result is found, the string `**ERROR**` is printed on the next line. For the above input, this is the output:

```
case 1: data is '[[[m,10,f],[f,7,m]],m,15,f]'
F----------MF-------M

case 2: data is '[[[m,10,f],[f,7,m]],m,15,m]'
Cannot connect

case 3: data is '[[[m,5,m],[f,20,f],[m,10,m]],f,10,f]'
M-----M
**ERROR**

case 4: data is '[[[m,5,m],[f,20,f],[m,10,m]],f,10,f]'
M-----MM----------M
**ERROR**

case 5: data is '[[[m,5,f]],f,10,f]'
M----------M
**ERROR**
```

Note that only successful connections need to be checked. If the result is "`Cannot connect`", no further analysis need be done.

This problem has several elements: You must extract cable set information, distance to span, and endpoint genders from the "case" line. You must extract cable configuration information from the result lines (composed of dashes, `M`s and `F`s). You'll need to be sure all the cables shown in the result are in the input set (i.e., be sure the programmer didn't fabricate a cable as shown in case 5). You'll need to be sure that all the inter-cable matings are proper and that the cable ends mate with the endpoints. You'll need to be sure that the cables span the distance.

You may assume that the input is well-formed. In particular you may assume that the first, fourth, seventh, etc. lines are "`case...`" lines and that the second, fifth, eighth, etc. lines are cable configurations. You may assume that all the cables in the cable configuration consist of an `M` or an `F` on each end and have one or more dashes in the middle. You may assume there are no extraneous characters present. For example, you may assume that you WON'T see something like this: `M---M--  ---M`.

A cable with given genders and length may appear more than once. For example, you might see this set of cables: `[[m,1,f],[m,1,f],[m,1,f]]`. If that set of cables were used to produce this configuration: `F-MF-MF-MF-M`, that would be invalid.

You may organize your solution in any way you wish, but here is a possible organization:

> (1) Write a routine `do_case(s)` that assumes `s` is a "`case...`" line and that returns a list that contains three values: A list of lists representing the cables, an integer representing the length to span, and a string that holds the desired endpoints.
>
> (2) Write a routine `do_config(s)` that processes a cable configuration line and produces a list of lists representing the cables.
>
> (3) Write a routine `check_sets(In,Out)` that takes two lists of cables and tests to be sure that the cables in `Out` are a subset of the cables in `In`.
>
> (4) Write a routine `check_config` to check other aspects of the configuration.

You may use any elements of Icon and you may also use `split`. You may assume you have a routine `ltos(L)` that takes a list and returns a string representation of its contents. For example, `ltos(["m",10,"f"])` would return "`m, 10, f`".

Don't forget to supply a main program that ties all the pieces together.

[Space for solution for problem 2]

[More space for solution for problem 2]

For the following Prolog problems you may assume you have at your disposal all the predicates discussed in class, such as `getone(Elem,List,Remaining)`, `length(List,Len)`, `last(Elem,List)`, *and* `append`. *You may also assume you have* `min(Elem,ListOfInts)` *and* `max(Elem,ListOfInts)`. *If there is a predicate you would like to use but you are in doubt as to its suitability, please ask.*

Problem 3 (8 points):

Write a Prolog predicate `sort(L,SortedL)` that describes the relationship that `SortedL` is a list that has the elements of `L` in ascending order. Both lists may be assumed to consist only of integers. The query `sort([],[])` should succeed.

```
| ?- sort([4,1,6,2],L).
L = [1,2,4,6] ? ;
no

| ?- sort([4,1,6,2],[1,2,4,6]).
yes

| ?- sort([4,1,6,2],[1,6,4,2]).
no

| ?- sort("just testing", S), name(A,S).

A = ' egijnsstttu',
S = [32,101,103,105,106,110,115,115,116,116,116,117] ?

yes
```

Regarding the last example, recall that a sequence of characters in double quotes is a shorthand for a list of integers representing the corresponding ASCII codes.

Problem 4 (6 points):

Write a Prolog predicate `hexprint/0` that prints, one per line, each of the hexadecimal numbers between `000` and `fff` inclusive but omitting those numbers where all the digits are the same (`000`, `111`, `222`, ..., `eee`, `fff`). In all, 4080 lines are to be printed (16^3 minus 16 is 4080). Note that `hexprint` ultimately succeeds.

Example:

```
| ?- hexprint.
001 (note that 000 was not printed)
002
003
004
005
006
007
[many lines omitted]
10d
10e
10f
110
112 (note that 111 was not printed)
113
[many many lines omitted]
ff9
ffa
ffb
ffc
ffd
ffe
     (note that fff was not printed)
yes
```

Problem 5 (6 points):

Write a Prolog predicate `palsum/1` that succeeds if a list of integer lists is palindromic with respect to the sums of the elements of the contained lists. Example:

```
| ?- palsum([[1,2], [5], [1,1,0,1]]).

yes
```

The above argument of `palsum` contains three lists whose sums are 3, 5, and 3, respectively. That list is said to be palindromic because the sequence of sums is the same whether read from left to right or right to left.

More examples:

```
| ?- palsum([[4], [5], [6], [3,1,1], [5,-1]]).

yes

| ?- palsum([[4], [15], [6], [3,1,1], [5,-1]]).

no

| ?- palsum([[4],[1,1,1,1]]).

yes

| ?- palsum([[0],[2],[3],[4],[3],[2],[]]).

yes

| ?- palsum([[1], [1,1], [2], [1,1,1]]).

no
```

Problem 6 (6 points):

It was said in class that a single Prolog predicate can take the place of several functions in some other language. (1) Explain what's meant by that statement. (2) Write a Prolog predicate that exhibits that property. (3) In some other language write a set of routines to perform the various operations that the predicate can perform. NOTE: Don't get carried away on this problem—conserve your time by using a simple predicate that illustrates the point.

Problem 7 (4 points):

Among ML, C++, Icon, and Prolog, which language did you find the most interesting, and why?

Problem 8 (4 points):

Among ML, C++, Icon, and Prolog, which language did you find the most difficult to learn. Cite an element of the language that you had particular difficulty with.

Problem 9 (3 points):

Write an ML function f of type `a' -> 'b -> 'a` that exhibits the following behavior:

```
- f;
val it = fn : 'a -> 'b -> 'a

- val f2 = f("one");
val f2 = fn : 'a -> string

- f2 2;
val it = "one" : string

- val f3 = f(10);
val f3 = fn : 'a -> int

- f3 20;
val it = 10 : int

- val g = f([1]);
val g = fn : 'a -> int list

- g [100];
val it = [1] : int list
```

Problem 10 (4 points):

Write an ML function `dups(s)` that removes sequences of duplicated characters from a string.

Example:

```
- dups;
val it = fn : string -> string

- dups("meet to eat beets");
val it = "met to eat bets" : string

- dups("....");
val it = "." : string

- dups("x");
val it = "x" : string

- dups("   a    test   right    here   ");
val it = " a test right here " : string

- dups("aaabbbcccdddeeeaaaabbb");
val it = "abcdeab" : string

- dups("");
val it = "" : string
```

Problem 11 (8 points):

In this problem you are to implement two C++ classes: `MList` and `PrintableMList`.

An `MList` holds a list of integer values that are added to the `MList` one by one via the < operator, which is overloaded.

To construct an `MList` the user must supply an integer value that specifies a maximum value for integers in the list. Attempts to add values larger than the limit are silently ignored.

An `MList` may be "closed" by calling its `close()` method. Once a list is closed it cannot be reopened. Attempts to add values to a closed `MList` are silently ignored.

The `size()` method returns an integer representing the number of values held in the `MList`.

Although there is no way to print an `MList` or obtain stored values from it, but the implementation of `MList` must properly maintain that information internally.

You may assume the user will never attempt to add more than fifty values to an `MList`.

`PrintableMList` is a derived class of `MList` that does everything an `MList` can do but can also print its contents in response to invocation of its `print()` method.

Here is a test program for `MList` and `PrintableMList`:

```
void main()
{
    MList m1(10);

    m1 < 5;     // Attempt to add some values to m1.  All should
    m1 < 7;     // go into m1 except for 20, which exceeds the
    m1 < 10;    // limit of 10.
    m1 < -10;
    m1 < 20;

    cout << "(1) m1.size() = " << m1.size() << endl;

    m1.close(); // Close m1 to prevent further additions

    m1 < 1;     // These additions should fail because m1
    m1 < 2;     // is closed.

    cout << "(2) m1.size() = " << m1.size() << endl;

                // m1 should now contain 5, 7, 10, and -10, but
                // there's no way to directly inspect that.

    PrintableMList m2(5);

    m2 < 1;
    m2 < 3;
    m2 < 6;     // 6 and 10 should bounce off because
    m2 < 10;    // they're greater than 5.

    m2.print();

    m2 < 2;
    m2 < 4;
    m2.close();
    m2 < 1;     // Should fail because m2 is closed.
    m2.print();
}
```

Output:

```
(1) m1.size() = 4
(2) m1.size() = 4
1 3          (output of first m2.print())
1 3 2 4      (output of second m2.print())
```

[Space for solution for problem 11]

EXTRA CREDIT SECTION

EC 1 (1 point):

Two of the programming languages mentioned during lectures that were first publicly released after 1990. Name BOTH of them.

EC 2 (1 point):

Reigning world chess champion Garry Kasparov was recently defeated in a six game match by IBM's Deep Blue system. What language was used to implement Deep Blue?

EC 3 (1 point):

Taking into account the syllabus, the slides for each language, homework assignments and solutions, and the mid-term exam and solutions, how many pages of handouts have been distributed in the course of this class? Your answer must be within 10% of the actual total. Note that a sheet printed on both sides counts as two pages.

EC 4 (1 point):

Name a language that is an ancestor of Icon.

EC 5 (1 point):

How many applications are included in the Prolog-1000 list? Pick one: (a) Less than 1000 (b) Exactly 1000 (c) More than 1000.

EC 6 (1 point):

What piece of sports equipment is on the instructor's desk?

EC 7 (1 point):

Write an Icon expression that is exactly ten characters in length and that evaluates to the value "10" (a string). RESTRICTION: You may use no letters, digits, underscores, white space characters, or parentheses.

EC 8 (1 point):

Finish this sentence as yourself: "If I only remember one thing from CSc 372 it will be _____."

EC 9 (1 point):

Without using a control structure (such as `every` or `while`) write an Icon expression that prints the letters from `"a"` to `"z"`.

EC 10 (1 point):

Write an Icon procedure `allsame(s)` that succeeds if all the characters in the string `s` are the same and fails otherwise. For example, `allsame("testing")` should fail, but `allsame("++++")` should succeed. `allsame("")` should fail.

CSc 372, Fall 2001
ML Examination
September 26, 2001

READ THIS FIRST

Do not turn this page until you are told to begin.

On problems that ask you only to write code, you need not include any explanation in your 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 have a question, raise your hand and the instructor or a teaching assistant will come to you. DO NOT leave your seat.

If you have a question that can be safely resolved by making an assumption, simply write down that assumption and proceed. Examples:

> "Assuming `length` is `string -> int`"
> "Assuming the homework function to insert values in a list is `insert_n`."
> "Assuming `ien(L,n,v)` and `repl(n,s)`" (if uncertain of argument order)

BE CAREFUL with assumptions that may significantly change a problem. For example, consider the expression `f [x 1, 2, 3]`. It may appear that a comma has been omitted in "`x 1`" but in fact there is no error.

You may use only language elements that have been studied in the class. You may use built-in functions such as `size`, `map`, and `rev`. You may use functions presented on the slides or written thereon, such as `member`, `filter` and `reduce`. You may use functions presented via the mailing list, such as `make_curried_r`. You may use functions that have appeared on the homework assignments this semester such as `gather` and `ien`. You may not use the `hd` or `tl` functions.

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

New DO NOT use your own paper. Do not write on the opposite side of the paper. If you need extra sheets, ask for them.

New Here is a list of functions that may be useful: `explode`, `filter`, `ien(L,n,v)`, `implode`, `map`, `print(s)`, `reduce`, `repl(s,n)`, `split d s`, `sum(L)`.

New There are no restrictions on the use of recursion and no deductions for poor style. Anything that works and doesn't use off-limits elements, such as `hd` and `tl`, is acceptable.

Print your name below **and when told to begin**, put your initials in the lower right hand corner of each page, being sure to check that you have all the pages.

This is a **forty-five** minute exam with a total of 100 points. There are nine problems on seven pages.

Seat row **and** number: _____ Name:_____

Problem 1: (2 points each; 8 points total)

State the ***type*** of each of the following expressions, or if the expression is not valid, state why. For example, the type of the expression `3+4` is `int` and the type of the expression `length` is `'a list -> int`.

```
(1, [2,3], 4.0)
```

```
(reduce op+)
```

```
explode o rev o implode
```

```
[[[size]]]
```

Problem 2: (3 points each; 15 points total)

State the ***type*** of each of the following functions:

```
fun a (w,h) = w * h * 1
```

```
fun f(x) = f(1) + f(2);
```

```
fun f(L,(a,b)) = L = (a,b)
```

```
fun f(3,4) = (size,length)
```

```
fun x y z = (y z) + z
```

Problem 3: (3 points each, 9 points total)

Edit or rewrite the following functions to make better use of the facilities of ML:

```
fun f(a,b,c) = [a-c, a+c]
```

```
fun f(x,y) = x::y::1::2::[]
```

```
fun f(n) = if n = 10 then true else if n = 5 then true else false;
```

Problem 4: (5 points)

Write the `map` function. The type of map is `('a -> 'b) -> 'a list -> 'b list`

Problem 5: (7 points)

Create a function `abslist(L)` of type `real list -> real list` that produces a copy of L with each value in the output list being the absolute value of the corresponding value in the input list. Assume there is NO function like Java's `Math.abs()` to compute absolute value–do the absolute value computation yourself.

Examples:

```
- abslist([1.0, ~2.0, ~3.4, 100.0]);
val it = [1.0, 2.0, 3.4, 100.0] : real list

- abslist([]);
val it = [] : real list
```

Problem 6: (7 points)

Your instructor suffered the great embarrassment of distributing a version of `gather` that has a bug: If called with an empty list it should return `[]` but in fact it returns `[[]]`. Example:

```
- gather([], 10);
val it = [[]] : int list list
```

Here is the source for the instructor's faulty version of `gather`:

```
fun gather(L, limit) =
  let
    fun g([], _, _, result) = [result]
     |  g(x::xs, sum, limit, result) =
         if result = [] then
            g(xs, x, limit, [x])
         else
            if x + sum > limit then
                result::g(x::xs, 0, limit, [])
            else
                g(xs, sum+x, limit, result @ [x])
  in
    g(L, 0, limit, [])
  end
```

Produce a version of `gather` that has the correct behavior. Either mark changes to the above code or rewrite it. Note: You only need to fix that one bug–don't search for additional problems. Your changes should not introduce additional bugs, of course.

Problem 7: (15 points)

In this problem you are to create TWO functions, `doubler` and `quadrupler`. `doubler` is of type `string list list -> string list list` and "doubles" each letter in the strings. `quadrupler` is of the same type, but quadruples each letter.

Examples:

```
- doubler([["just"], ["a", "test"], ["h", "e", "r", "e", ""]]);
val it = [["jjuusstt"],["aa","tteesstt"],["hh","ee","rr","ee",""]]
  : string list list

- doubler([["hello"]]);
val it = [["hheellllloo"]] : string list list

- doubler([ ]);
val it = [] : string list list

- doubler([[ ]]);
val it = [[]] : string list list

- quadrupler([["an", "example"],[ ]]);
val it = [["aaaannnn","eeeexxxxaaaammmmpppplllleeee"],[]] : string
list list
```

Problem 8a: (7 points)

Create a function `genlist` that takes a list of integers and for each integer N in the list, produces a list with N instances of the number 1. You may assume that all the values are non-negative.

Examples:

```
- genlist;
val it = fn : int list -> int list list
- genlist [2,1,0,7,4];
val it = [[1,1],[1],[],[1,1,1,1,1,1,1],[1,1,1,1]] : int list list
- genlist [];
val it = [] : int list list
- genlist [10];
val it = [[1,1,1,1,1,1,1,1,1,1]] : int list list
```

Problem 8b: (7 points)

Create a function `genlist_inv` that performs the inverse operation of `genlist`. The only value appearing in the lists will be the integer 1 (one). Examples:

```
- genlist_inv [[1,1,1], [1], [1,1]];
val it = [3,1,2] : int list
- genlist_inv (genlist [2,1,0,7,4]);
val it = [2,1,0,7,4] : int list
- genlist_inv [[],[],[1],[]];
val it = [0,0,1,0] : int list
```

Problem 8c: (2 points)

What is the type of `genlist_inv o genlist o genlist_inv` ?

Problem 9: (18 points)

Imagine a function `read_all_bytes(fname)` that produces a string containing all the bytes in the file `fname`. For a file "x" containing the following five lines of text,

```
this
is @ some data
here to
test@
with
```

the string `"this\nis @ some data\nhere to\ntest@\nwith\n"` is returned by `read_all_bytes("x")`. Note that `\n` represents one character, a newline.

Create a function `tacdel(fname)` that reads the file named by `fname` and prints (using the `print` function) the lines in the file in reverse order, and if a line contains the character "`@`", the line "`<D>`" appears in its place. Assume that the file exists and is readable. Example:

```
- tacdel("x");
with
<D>
here to
<D>
this
val it = () : unit
```

Here is a function you may use:

```
fun member (v, []) = false
  |  member (v,x::xs) = if x = v then true else member(v,xs)
```

Name:_____ Seat row **and** number: _____

CSc 372, Fall 2001
Icon Examination
October 24, 2001

READ THIS FIRST

Fill in your name and seat row/number above.

Do not turn this page until you are told to begin.

DO NOT use your own paper. Do not write on the opposite side of the paper. If you need extra sheets, ask for them.

If you have a question that can be safely resolved by making an assumption, simply write down that assumption and proceed. Examples:

> "Assuming `reverse(s)` reverses a string"
> "Assuming `*t` returns the number of keys in table `t`"

Changed If you have a question you wish to ask, raise your hand and the instructor or teaching assistant will come to you. DO NOT leave your seat.

You may use all elements of Icon regardless of whether they have been studied in class. You may not use any elements of the Icon Program Library (IPL). (The instructor has said nothing about anything in the IPL during lectures.)

Changed You may use Icon procedures that appear on the slides, or have been presented in class, or have been mentioned in e-mail. Here are some Icon procedures that may be useful: `split`, `atos`, `rev`, and `read_file`

There are no deductions for poor style. Anything that works and meets all restrictions will be worth full credit, but try to keep your solutions brief to save time.

On problems that ask you only to write code, you need not include any explanation in your answer if you are confident it is correct. However, if an answer is incorrect, any accompanying explanation may help you earn partial credit.

This is a **forty-five** minute exam with a total of 100 points and eight possible points of extra credit. There are eight regular problems and six extra credit problems.

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

When told to begin, double-check that your name and seat/row are recorded at the top of this page, and then put your initials in the lower right hand corner of each page, being sure to check that you have all the pages.

Problem 1: (15 points)

Write a program `tacdel` that reads lines from standard input and prints the lines in the file in reverse order. If a line contains the character "@", the line "<D>" appears in place of that line. Example:

```
% cat x
this
is @ some data
here to
test@
with
% tacdel < x
with
<D>
here to
<D>
this
%
```

Problem 2: (15 points)

Write a program `idiff` that examines two files named on the command line and if the files are not identical, prints "`Diffs`". If the files are identical, `idiff` produces no output.

Two files are considered to be identical if and only if they contain the same number of lines and the lines in each are identical and in the same order. Here are four sample input files. Note that `f1` and `f3` are identical.

% **cat f1**	% **cat f2**	% **cat f3**	% **cat f4**
a	here	a	1
test	is	test	2
here	a	here	3
%	test	%	%
	%		

Examples of usage:

```
% idiff f1 f2
Diffs
% idiff f1 f3
% idiff f2 f4
Diffs
% idiff f1 f1
%
```

Problem 3: (15 points)

Write a program `paldate` that searches for palindromic dates between January 1, 2001 and December 31, 2099 and prints those dates. Represent a date such March 15, 2001 like this: `3/15/1`.

Some examples of palindromic dates are `1/1/1`, `1/22/1`, `3/11/3`, and `12/11/21`.

Note that months 4, 6, 9, 11 (April, June, September, and November) have 30 days and assume that February always has 28 days. All the other months have 31 days.

RESTRICTION: You may not use any lists in your program.

The output of `paldate` is always the same:

```
% paldate
1/1/1
1/2/1
1/3/1
1/4/1
1/5/1
1/6/1
1/7/1
1/8/1
1/9/1
1/11/1
...lots more lines...
```

Problem 4: (24 points)

Write a program `total` that reads merchandise descriptions and prices and then computes the total for a list of items to purchase.

Both merchandise descriptions and the list to purchase are read from standard input. One or more lines of merchandise item names and associated prices appear first, one per line. A line containing only a "`.`" signals the end of the descriptions, and that the full names of items to purchase follow. The output of the program is the total price of all the items listed after the end of the descriptions.

Here is a sample input file:

```
% cat total.in
Icon book                    $10
Old lamp with bad switch      $2
Bowling ball                  $1
Perl book                     2c
Pencil                        6c
.
Pencil
Bowling ball
Pencil
Icon book
Old lamp with bad switch
%
```

The program outputs the total cost of the to-purchase items. Example:

```
% total < total.in
$13.12
%
```

Details:

Prices will be something like `$15` (fifteen dollars) or `10c` (ten cents). Prices are always integer amounts—there won't be a price like `$10.15`. A price is always preceded by a dollar sign or followed by a "`c`". Prices appear at the very end of a line—the last character of the first line of input is "`0`".

The whitespace characters are always blanks, never tabs. In the "to-purchase" list, no blanks follow the item name.

Don't worry about formatting—just print the total preceded by a dollar sign.

Assume that there is no invalid input, no duplications in the price list, and that all items named in the to-purchase list have appeared in the price list.

(Problem 4 — space for answer)

For reference: (copied from previous page)

```
% cat total.in
Icon book                    $10
Old lamp with bad switch      $2
Bowling ball                  $1
Perl book                     2c
Pencil                        6c
.
Pencil
Bowling ball
Pencil
Icon book
Old lamp with bad switch
% total < total.in
$13.12
%
```

Problem 5: (7 points)

Write a procedure `intmem(i, L)` that returns `&null` if the integer `i` is contained in the list `L` and fails otherwise. `L` may contain values of any type.

Examples:

```
][ intmem(1, [3, 1, 4, 2, 5]);
   r := &null  (null)

][ intmem(1, [3, "abc", &digits, 1, []]);
   r := &null  (null)

][ intmem(1, [3, "abc", &digits, 10, []]);
Failure

][ intmem(1, []);
Failure
```

Only the top-level elements of `L` need be considered. Do not search for `i` in lists contained in `L`:

```
][ intmem(1, [[1]]);
Failure
```

Note that a comparison such as `1 = []` produces a run-time error.

Problem 6: (10 points)

Write a program `sumints` that reads lines on standard input and prints the sum of all integers found.

Example:

```
% sumints
On February 14, 1912, Arizona became
the 48th state.
^D (control-D)
1974
%
```

A string such as `12.34` is simply interpreted as two integers: 12 and 34.

Restriction: Your solution must be based on string scanning. The only types you may use are integers, strings, and character sets. You may not any comparison operators such as ==.

Problem 7: (6 points)

According to the instructor, what is the unique aspect of Icon's expression evaluation mechanism?

Name one thing that the instructor doesn't like about Icon or has identified as a problem with the language. Here's one thing you can't mention: unexpected failure.

There are no built-in functions in Icon to do things like reverse lists, compare all elements of two lists, or do a "deep copy" of a list. What did the instructor cite as the likely reason for the absence of functions like that?

Problem 8: (8 points)

Fill in the blanks:

The instructor described the function _____ as being a "lone wolf". He said that _____ "works well with others".

The result of a successful comparison in Icon is _____.

We studied a total of ____ string scanning functions. Of those, two changed _____

and _____ of them returned a _____. _____ is one-of-a-kind.

EXTRA CREDIT SECTION (one point each)

(a) Write the result sequence of `(?"xxx" || !"xxx" || *"xxx")`

(b) If the string `s` contains your login name, what is `string(cset(s[1:4]))[1:-2]`?

(c) Which one of the following principal contributors to Icon have not been mentioned in class: Steve Wampler, Bob Alexander or Tim Korb?

(d) Cite up to three elements of Icon that are "syntactic sugar". (one point each)

(e) Given this program, `args.icn`:

```
procedure main(args)
    every write(!args)
end
```

what does `args` print when run like this: "`args < in.dat >out.dat`"?

(f) Describe a situation where `tab(n)` and `move(n)` produce the same result, for a particular subject, position, and value of `n`.

Name:_____ Seat row **and** number: _____

CSc 372, Fall 2001
Prolog Examination
November 28, 2001

READ THIS FIRST

Fill in your name and seat row/number above.

Do not turn this page until you are told to begin.

DO NOT use your own paper. Do not write on the opposite side of the paper. If you need extra sheets, ask for them.

If you have a question that can be safely resolved by making an assumption, simply write down that assumption and proceed. Examples:

> "Assuming `length(List,Length)`"
> "Assuming `3 / 4` produces a floating point result"

If you have a question you wish to ask, raise your hand and the instructor or teaching assistant will come to you. DO NOT leave your seat.

You may use only language elements and predicates that have been studied in class or mentioned in mail. Here are some predicates that may be useful: `member(X, List)`, `sum(List, Sum)`, `length(List, Length)`, `append(L1,L2,L3)`, `getone(Element, List, Remaining)`.

There are no deductions for poor style. Anything that works and meets all restrictions will be worth full credit, but try to keep your solutions brief to save time.

Unless otherwise specified, predicates should produce only one result.

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.

This is a **forty-five** minute exam with a total of 100 points and eight possible points of extra credit. There are eleven regular problems and an extra credit section with eight problems.

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

When told to begin, double-check that your name and seat/row are recorded at the top of this page, and then put your initials in the lower right hand corner of each page, being sure to check that you have all the pages.

Problem 1: (5 points)

Show an example of each of the following:

A fact:

A rule:

A query:

A clause:

An atom:

Problem 2: (2 points)

What is the relationship between facts, rules, and clauses?

Problem 3: (5 points)

True or false: The following is a working implementation of the `member` predicate, as studied in class:

```
member(X, [X]).
member(X, [_|T]) :- member(X, T).
```

Problem 4: (5 points)

True or false: The following is a working implementation of the `length` predicate, as studied in class:

```
length([], 0).
length([_|T], Sum) :- length(T, Sum), Sum is Sum + 1.
```

Problem 5: (12 points)

Write a predicate `sumval(+List, +Value, -Sum)` that produces the sum of all occurrences of `Value` in the list `List`. Assume that `Value` and all elements in `List` are integers. `sumval` should produce only one result. You may abbreviate `sumval` as `sv`.

```
| ?- sumval([3, 2, 1, 1, 3, 1], 1, Sum).
Sum = 3 ?

| ?- sumval([3, 2, 1, 1, 3, 1], 3, Sum).
Sum = 6 ?

| ?- sumval([3, 2, 1, 1, 3, 1], 5, Sum).
Sum = 0 ?

| ?- sumval([ ], 10, Sum).
Sum = 0 ?
```

Problem 6: (4 points)

Write a predicate `sumvals(+List, +ListOfValues, -Sum)` that produces the sum of all occurrences of members of the list `ListOfValues` in the list `List`. Assume that `Value` and all elements in `List` are integers. `sumvals` should produce only one result. You may abbreviate `sumvals` as `svs`.

Note that the only difference between `sumval` (the previous problem) and `sumvals` is that `sumvals` has a list of values that are to be included in the sum..

```
| ?- sumvals([3,2,1,1,3,1], [1], Sum).
Sum = 3 ?

| ?- sumvals([3,2,1,1,3,1], [1,3], Sum).
Sum = 9 ?

| ?- sumvals([3,2,1,1,3,1], [1,3,2], Sum).
Sum = 11 ?

| ?- sumvals([3,2,1,1,3,1], [-1,-3,-2], Sum).
Sum = 0 ?

| ?- sumvals([3,2,1,1,3,1], [], Sum).
Sum = 0 ?

| ?- sumvals([], [], Sum).
Sum = 0 ?

| ?- sumval2([3,2,1,1,3,1], [3,1,1,3,1,3], Sum).
Sum = 9 ?
```

The order of values in `ListOfValues` is inconsequential. Note that a given value may appear multiple times in `ListOfValues` but that does not affect the result.

Problem 7: (12 points)

Write a predicate `listeq(+L1, +L2)` that succeeds if the lists `L1` and `L2` are identical and fails otherwise. `L1` and `L2` may be arbitrarily complicated lists but all values will be either integers or lists.

```
| ?- listeq([1,2,3],[1,2,3]).
yes

| ?- listeq([1,2,3],[1,2,3,4]).
no

| ?- listeq([1,[2,[],[3,4,5],[6],8],  [1,[2,[],[3,4,5],[6],8]).
yes

| ?- listeq([1,[2,[],[3,4,5],[6,[7]]],8],  [1,[2,[],[]]]).
no

| ?- listeq([],[]).
yes

| ?- listeq([],[[]]).
no
```

The instructor believes there is a significant chance of making a careless error in the answer for `listeq` that may result in a major deduction. BE CAREFUL AND DOUBLE-CHECK YOUR ANSWER!!

Problem 8: (15 points)

Write a predicate `consec(+Value, +N, +List)` that succeeds if and only if `List` contains `N` consecutive occurrences of `Value`. Assume that `Value` and all elements of `List` are atoms or integers. Assume `N > 0`.

```
| ?- consec(a, 2, [b,c,a,a,b,a]).
yes

| ?- consec(a, 2, [b,c,a,b,a]).
no

| ?- consec(a, 1, [b,c,a,b,a]).
yes

| ?- consec(b, 3, [b,b,c,b,bb,bbb,a,b,a]).
no

| ?- consec(b, 3, [b,b,c,b,b,b,a,b,a]).
yes

| ?- consec(1, 4, [a,b,c,d,1,2,3,4,1,1,1,1]).
yes
```

For this problem you may assume that you have a `repl(X, N, List)` predicate:

```
| ?- repl(b,3,L).
L = [b,b,b] ?
```

Problem 9: (15 points)

Write a predicate `order3(+L1, -L2)` that assumes that `L1` contains three integers and instantiates `L2` to be a list of those integers in ascending order:

```
| ?- order3([3,1,4],L).
L = [1,3,4] ?

| ?- order3([4,1,3],L).
L = [1,3,4] ?

| ?- order3([4,-1,3],L).
L = [-1,3,4] ?

| ?- order3([1,1,1],L).
L = [1,1,1] ?
```

Note: The instructor's solution uses `getone(X, L, Remaining).`

Problem 10: (20 points)

In this problem you are to write a predicate `inventory/0` that does an inventory calculation for a fruit stand. There are instances of `fruit/1` and `cost/2` for each type of fruit that the stand may stock. There are instances of `qty/2` (quantity) for each type of fruit that is currently in stock.

Example:

`fruit(apple).`
`fruit(banana).`
`fruit(orange).`
`fruit(pear).`
	`cost(apple, 30).`
	`cost(pear, 50).`
	`cost(banana, 15).`
	`cost(orange, 25).`
		`qty(apple, 3).`
		`qty(orange, 5).`

There four types of fruit, and the cost of each is expressed in cents. There are three apples and five oranges currently in stock. For the above collection of facts, `inventory` does this:

```
| ?- inventory.
apple: 3 at 30 = $0.90
banana: 0 at 15 = $0.00
orange: 5 at 25 = $1.25
pear: 0 at 50 = $0.00

yes
```

Note that `inventory` lists the fruits in the order the `fruit` facts appear. The elements in each line of output are separated only by spaces and thus the output text weaves back and forth. Note that `inventory` always succeeds.

There will be at least one fruit and each fruit fact will have an associated cost fact present. At least one fruit will be in stock. (That is, there will be at least one `qty` fact.) Assume that printing a floating point number with `write` or `format` produces two places to the right of the decimal point—exactly what's needed.

Problem 11: (5 points)

For each of the two following queries, write in the values computed for each variable. If a query fails, indicate it.

```
| ?- X = [1,2,3], Y = [4|X], [A,B|C] = Y.

A =

B =

C =
```

```
| ?- A = [], B = 1, C = [A,B], B = 2, [D,E] = C.

D =

E =
```

EXTRA CREDIT SECTION (one point each)

(a) What is the Prolog 1000?

(b) In what country was Prolog developed?

(c) What country made a big investment in Prolog?

(d) What language was used for the first implementation of Prolog?

(e) What is the sound of a combinatorial explosion?

(f) What is inaccurate about this specification: `append(+L1,+L2,-L3)`?

(g) Why is a warning about a singleton variable significant?

(h) Recall that the ML functions `ord` and `chr` convert between ASCII character codes and ASCII characters. In a Prolog library you see these two predicates: `get_chr(+Number, -Char)` and `get_ord(+Char, -Number)`. What's odd about that?

Name:_____ Seat row **and** number: _____

CSc 372, Fall 2001
Final Examination
December 14, 2001

READ THIS FIRST

Fill in your name and seat row/number above.

Do not turn this page until you are told to begin.

DO NOT use your own paper. DO NOT write on the opposite side of the paper. If you need extra sheets, ask for them.

If you have a question that can be safely resolved by making an assumption, simply write down that assumption and proceed.

If you have a question you wish to ask, raise your hand and the instructor or teaching assistant will come to you. DO NOT leave your seat.

For the problems that ask you to write Emacs Lisp or Icon code, you may use all elements of the language and libraries, whether or not they were covered in class. For the ML and Prolog problems you may use only the elements of the language and libraries that were studied in class.

It is tedious to properly match parentheses when writing Lisp code on paper. You may draw brackets to indicate the intended grouping of your Lisp code.

Aside from problem 7, there are no deductions for poor style. Anything that works and meets all restrictions will be worth full credit, but try to keep your solutions brief to save time.

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.

This is a **110** minute exam with a total of 100 points and nine possible points of extra credit. There are twelve regular problems and an extra credit section with six problems.

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

When told to begin, double-check that your name and seat/row are recorded at the top of this page then put your initials in the lower right-hand corner of each page, being sure to check that you have all the pages.

Quick reference for some possibly useful Emacs Lisp functions
(You may tear this page out)

`(char-after` *N*`)`	Returns the character after the Nth position in the buffer.
`(delete-char` *N*`)`	Deletes N characters following the point.
`(delete-region B E)`	Deletes text between buffer positions B and E.
`(eobp)`	Tests whether the point is at the end of the buffer.
`(forward-line` *N*`)`	Moves to the beginning of the Nth line relative to the current line. N may be zero or negative.
`(get-empty-buffer B)`	Creates an empty buffer named B.
`(goto-char N)`	Sets the point to position N.
`(int-to-string` *N*`)`	Converts the integer N to a string.
`(not E)`	Returns `t` if E is `nil`; returns `nil` otherwise.
`(sit-for 0 N)`	Pauses for N milliseconds.
`(switch-to-buffer B)`	Switches to the buffer named B.
`(window-height)`	Returns the height of the current window.
`(window-width)`	Returns the width of the current window.

Problem 1: (7 points)

Write an Emacs Lisp function `change` that changes every letter in the current buffer to "`x`", and every decimal digit to "`#`" . You may assume the presence of a function `letterp` that returns `t` if the function's argument is a letter and nil otherwise. For example, `(letterp ?A)` returns `t`. Assume a similar function, `digitp`, to test if a character is a decimal digit.

These buffer contents:

```
On February 14, 1912, Arizona became
the 48th state.
```

would be changed to this:

```
xx xxxxxxxx ##, ####, xxxxxxx xxxxxx
xxx ##xx xxxxx.
```

IMPORTANT: Select and solve two of the next three problems (problems 2, 3, and 4). If you do all three, the first two will be graded. Problem 2 carries a three point extra credit bonus.

Problem 2: (15 points, plus 3 of extra credit—do only two of problems 2, 3, and 4)

When editing text it is sometimes useful to know the column in which certain text appears. In this problem you are to write an Emacs Lisp function named `ruler`, bound to `ESC-R`, that inserts a "ruler" before the current line. The ruler disappears when the user strikes any key and the position of the point is restored to what it was initially.

For example, consider the following buffer contents and imagine that the cursor is positioned on the "`e`" in "`while`".

```
(goto-char 1)
(while (not (= (point) (point-max)))
    (setq char (char-after (point)))
```

The user then types ESC-R and a two-line ruler with column positions is inserted in the text before the current line:

```
(goto-char 1)
     1         2         3         4         5         6
1234567890123456789012345678901234567890123456789012345678901234567890123456789012345
(while (not (= (point) (point-max)))
    (setq char (char-after (point)))
```

We can see that the "`w`" is in column six and the last parenthesis is in column forty. **The ruler should be as wide as the window, which can be assumed to be less than 100 characters in width.**

When the user types any character, the text of the ruler is removed and the buffer is restored to its original state, with the cursor positioned on the "`e`" in "`while`":

```
(goto-char 1)
(while (not (= (point) (point-max)))
    (setq char (char-after (point)))
```

Note that the ruler is created by inserting text in the buffer. `(read-char)` is then called to read a character of input and when `read-char` returns, the inserted text is deleted.

A blank page follows.

(Space for solution for problem 2)

For reference:

```
          1         2         3         4         5         6
12345678901234567890123456789012345678901234567890123456789012345
```

Problem 3: (15 points—do only two of problems 2, 3, and 4)

Write an Emacs Lisp function `longest` that positions the cursor at the beginning of the longest line in the current buffer. If there are ties for the longest line, choose the first one.

Problem 4: (15 points—do only two of problems 2, 3, and 4)

Write an Emacs Lisp function `marquee` that displays a horizontally scrolling line of text. (There is a standard Windows screen saver having the same name.) The user is prompted for the text to display with `Text?`. The text is displayed at the left side of the center line of the window. Every 70 milliseconds, the leftmost character is removed and put at the right end of the string. This creates a perception of the text moving from right to left. The text is displayed in a buffer named "`*m*`".

For the text "`CSc 372 Final Examination...`" the line being displayed would first be this:

```
CSc 372 Final Examination...
```

then this:

```
Sc 372 Final Examination...C
```

then this:

```
c 372 Final Examination...CS
```

then this:

```
 372 Final Examination...CSc
```

and so forth.

`marquee` never terminates on its own. It runs until the user terminates it.

A blank page follows.

(Space for solution for problem 4)

Problem 5: (9 points)

Background: In Icon, built-in functions supply default values for omitted arguments when there's a reasonable default. Emacs Lisp is very inconsistent in this regard. For example, the count for `forward-line` is optional but the count for `delete-char` is required. The instructor believes that consistent use of reasonable defaults would be an improvement for Emacs Lisp.

Problem: Identify two more elements from ML, Icon, Prolog, and/or Java that would be improvements for Emacs Lisp. Briefly describe how Emacs Lisp would benefit from their inclusion. The elements may be from the same language or from different languages.

Problem 6: (6 points)

Write an ML function `rm_prefix(P, L)` that **ASSUMES** that P is a prefix of the list L and returns a copy of L with P removed.

```
- rm_prefix([1,2],[1,2,3,4]);
val it = [3,4] : int list

- rm_prefix([ ], [5,10,20]);
val it = [5,10,20] : int list

- rm_prefix(explode "test", explode "testing");
val it = [#"i",#"n",#"g"] : char list

- implode(rm_prefix(explode "test", explode "testing"));
val it = "ing" : string
```

Be sure to keep in mind that `rm_prefix` **ASSUMES** that `P` is a prefix of `L`:

```
- rm_prefix([1,2], [3,4,5]);
val it = [5] : int list

- rm_prefix([1,2,3], explode "testing");
val it = [#"t",#"i",#"n",#"g"] : char list

- rm_prefix([1.0,2.0],[5,4,3,2,1]);
val it = [3,2,1] : int list
```

Problem 7: (2 points)

Write an ML function `listeq(L1, L2)` that returns true if the lists `L1` and `L2` are identical and returns false otherwise. Style does matter on this problem.

```
- listeq;
val it = fn : ''a * ''a -> bool

- listeq([1,2,3],[1,2,3]);
val it = true : bool

- listeq(explode "testing", explode "test");
val it = false : bool

- listeq([[],[],[]], [[],[],[]]);
val it = true : bool
```

Problem 8: (8 points)

Write an Icon procedure `extsort(L)` that accepts a list of file names and sorts them by their extension.

```
][ extsort(["x.icn","test.c","b.java","a.c","a.java"]);
   r := ["a.c","test.c","x.icn","a.java","b.java"]  (list)
```

You may assume that file names contain exactly one dot. Files with the same extension may appear in any order. (In the above, for example, it would be acceptable for "`test.c`" to precede "`a.c`".)

Problem 9: (8 points)

Write a Prolog predicate `find(+N, [-A,-B,-C])` that produces all combinations of integers between 1 and `N` inclusive such that `N = A * B - C`. You may assume that you have a predicate `iota(+N,-L)` that instantiates `L` to a list of the integers from 1 through `N`.

Examples:

```
| ?- find(5,R).
R = [2,3,1] ? ;
R = [2,4,3] ? ;
R = [3,2,1] ? ;
R = [4,2,3] ? ;
no
```

```
| ?- find(4,R).
no
```

```
| ?- find(100,R).
R = [2,52,4] ? ;
R = [2,53,6] ? ;
R = [2,54,8] ? ;
...lots more...
```

Important: Each integer may appear only once in a given result. There should never be a result like `R = [5,5,5]` or `R = [7,1,7]`.

Hint: This is the instructor's second and final attempt (no pun intended) to get everybody to use `getone(X, L, R)` on a test.

Problem 10: (10 points)

Pick one interesting element from any of the languages we have studied and, in the style of an e-mail note, describe that element to a colleague familiar only with Java programming. Be sure to talk about the syntactic form, the operation, and describe a practical usage of the element.

Problem 11: (10 points)

There have been two notable attempts to produce a single language to meet the needs of the majority of programmers. In the 1960s, the language PL/I was created to serve both the business and scientific programming communities. In the 1970s, the Department of Defense sponsored the development of the Ada programming language, which was to be used for DoD software systems whenever possible.

Present an argument either for or against the prospect of a single language becoming dominant for the majority of software development. Here are some points to possibly consider: What major elements would a dominant language need to have? How might such a language come into existence? What are forces that would work in opposition to a language becoming dominant?

Problem 12: (10 points)

For five points each, answer TWO of the following questions. If more than two questions are answered, only the first answers on the paper will be graded.

(A) In languages like Java, C++, and ML, words like "if", "fun", and "class" are called *reserved words*—they can't be used for variable or routine names. The PL/I language has no reserved words. One can write statements like `if = while + then(else)`. What are some advantages and/or disadvantages to having no reserved words? Disregard issues related to compiling languages with no reserved words.

(B) Icon has several polymorphic operators and functions. For example, the unary `*` operator returns the number of elements in an object. The `delete` function removes elements from sets and tables. What are some tradeoffs that a language designer must take into account when considering inclusion of a polymorphic operation in a language?

(C) Imagine a language that has heterogeneous lists like Prolog, Icon, and Lisp, but that also has a tuple type that is very similar to ML. Present an argument either for or against the claim that if a language has heterogeneous lists, tuples provide no additional benefit.

(D) What do Java's exception handling mechanism and Icon's failure mechanism have in common?

(E) The instructor said that "every programmer should have a language like Icon in their toolbox". What are the characteristics of Icon, and similar languages, that make it worthwhile to know such a language in addition to mainstream languages like Java and C++? (Or, argue that knowing a single language like Java or C++, and knowing it well, is all that's needed.)

(F) A language implemented in C, such as Emacs Lisp, typically has some library functions coded in C and the balance coded in the language itself. What factors influence whether a function is implemented in the language itself?

A blank page follows.

(Space for answers for problem 12)

EXTRA CREDIT SECTION (one point each)

(a) What's the difference between a hack and programming technique?

(b) Order these languages from oldest to youngest: Icon, Java, Lisp, ML, Prolog.

(c) Write an expression that is valid in three of the languages we studied but with a notably different interpretation in each.

(d) As of midnight on December 12, how many messages have been sent to the CSc 372 mailing list? (Your answer must be within 10%.)

(e) The instructor has immediate plans to rewrite the Icon assignments in another language to see how that language compares to Icon. What's the language?

(f) What is the type of `rm_prefix` (problem 6)?

CS login: _____ Seat Number: _____

CSc 372 Mid-Term Examination
October 17, 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 60-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!

There is an exam-wide restriction: You may not use regular expressions or hashes in a Ruby solution.

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

> Assuming `String.sub` is `string * int -> char`
> Assuming `tl []` returns `[]`.
> 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.

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 12 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: (5 points)

The instructor often says "In ML we never change anything; we only make new things." What does he mean by that?

Problem 2: (5 points)

(a) Write an ML function `fun firstLast(L)` that returns a tuple consisting of the the first and last elements of the list `L`. Assume `L` has at least one element. **Restriction: You may use helper functions but you may use no functions other than functions you write.** Hint: Write a helper function `last(L)`.

(b) What is the type of `firstLast`?

Problem 3: (5 points)

Consider a list of one-element lists:

```
[[10], [2], [5], [77]]
```

Imagine an ML function `f` that will "flatten" such lists, like this:

```
- f [[10], [2], [5], [77]];
 val it = [10,2,5,77] : int list
```

Using at most eight (8) characters, fill in the blank below to create `f`:

```
val f = _ _ _ _ _ _ _ _
```

Problem 4: (6 points)

Write an ML function `eo(L)` that returns a list consisting of every other element of the list `L`. (That is, the 2nd, 4th, 6th, 8th, ... elements.) **Restriction: You may use helper functions but you may use no functions other than functions you write.** Examples:

```
- eo [1,7,9,12];
val it = [7,12] : int list

- eo [5,3,10];
val it = [3] : int list

- eo (iota 10);
val it = [2,4,6,8,10] : int list
```

Problem 5: (12 points)

Without writing a function that is directly or indirectly recursive, write an ML function `ints_to_string(L)` that produces a string representation of `L`, an `int list`. As shown below, the values are separated by a comma and a space. If you wish, you may use `Int.toString`, of type `int -> string`, to convert individual values. (It produces `"~3"` for `~3`.) Examples:

```
- ints_to_string([10,5,3,~3]);
val it = "10, 5, 3, ~3" : string

- ints_to_string([123]);
val it = "123" : string

- ints_to_string([]);
val it = "" : string
```

Problem 6: (15 points)

Without writing a function that is directly or indirectly recursive, write an ML function `show_lists(L)` of type `(string * int list) list -> unit` that prints the contents of `L`, prefixing each `int list` with the specified label. Example:

```
- show_lists [("a",[3,5,1]),("list2",[]),("c",(iota 10))];
a: 3, 5, 1
list2: <empty>
c: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
val it = () : unit

- show_lists [("Result(s)", [10])];
Result(s): 10
val it = () : unit

- show_lists [];
val it = () : unit
```

Note that if a tuple's `int list` is empty, the string `"<empty>"` is printed. Be sure that your solution does not produce any trailing whitespace—the last digit, or `"<empty>"`, should be immediately followed by a newline.

If you wish, you may make use of `ints_to_string`, from the previous problem. (Assume a working version.)

Problem 7: (10 points)

Consider a folding that operates on an `int list` with an even number of values, all greater than zero, and produces a list of pair-wise sums: [*1st+2nd, 3rd+4th, ..., (N-1)th+Nth*]

Example:

```
- foldr f [] [5,7, 3,4, 9,8];
val it = [12,7,17] : int list

- foldr f [] [10,20];
val it = [30] : int list

- foldr f [] [4,1, 3,2, 2,3, 4,1];
val it = [5,5,5,5] : int list
```

In this problem you are to write a function `f` such that the above foldings work as shown.

Hint: Remember the technique of writing out a fixed expansion of the calls, like `f(e1,f(e2,f(e3,[])))`, to help develop the function to fold with.

Keep in mind that you are writing a function to use with `foldr`, not a function that performs this pair-wise summing directly.

Hint: I think that to create an exam-size solution you must take advantage of the fact that all values are greater than zero. If you don't quickly see an approach, you may be wise to skip this problem and come back to it if time permits.

Problem 8: (4 points)

(a) Is the following ML function declaration valid? If valid, what is the type of `f1`? If not valid, explain why it is not valid.

```
fun f1 () f2 () = [f2];
```

(b) Write an ML function `g` whose type is

```
int -> (int list * int) -> (string list) -> (bool * real)
```

UNLIKE `ftypes.sml` on the first assignment, there are no restrictions on this problem.

Problem 9: (4 points)

(a) Using nothing but a `val` binding and the composition operator, create an ML function `f(s)` that returns a copy of the string `s` with the first and last characters removed. Assume that `size(s) >= 2`.

Examples:

```
- f "string";
val it = "trin" : string

- f "ab";
val it = "" : string

- f "abc";
val it = "b" : string
```

The requirements imply that your solution must look like this:

```
val f = f1 o f2 o ... o fN.
```

(b) As you've defined it, what is the type of `f`? (Don't forget to properly account for the intermediate functions in the composition.)

Problem 10: (16 points)

When showing examples of interaction with `irb` it saves space to put both the expression and the result on the same line. However, it is tedious to left-align the results in a column.

Write a Ruby program that reads from standard input a script of interaction with `irb` and combines the expressions and results on a single line, vertically aligning the results, based on the longest input expression. Each combined line is followed by a blank-line.

An optional command line argument specifies the number of spaces between the end of the longest input expression and its result. (Assume the specified spacing is good, not `"x"`, `"5x"` or `"-3"`, for example.) If no argument is specified, one space is used. Example:

```
% cat irbfmt.1
>> 3+4
=> 7
>> a = %w{words in array}
=> ["words", "in", "array"]
>> a.max
=> "words"
% ruby irbfmt.rb 3 < irbfmt.1   # NOTE: 3 spaces before "=>"
>> 3+4                      => 7

>> a = %w{words in array}   => ["words", "in", "array"]

>> a.max                    => "words"

%
```

Because expressions and results should always be paired, it is an error if the number of input lines is odd:

```
% cat irbfmt.2
>> 3*7
=> 21
>> it.class
% ruby irbfmt.rb < irbfmt.2
Error: short input
```

Write your answer below or use the whole page that follows.

Don't forget to handle the optional command-line argument.

(Space for `irbfmt.rb`)

Problem 11: (2 points)

Write a Ruby program that reads all lines from standard input and prints them on standard output in reverse order, last line first, first line last. Assume there is at least one line and that the file ends with a newline.

```
% cat revlines.1
reverse
the
order
% ruby revlines.rb < revlines.1
order
the
reverse
%
```

For two points of extra credit, have less than 25 characters in your solution. (No abbreviations on this one!) To help you pursue this option, here are 24 blanks:

```
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

Problem 12: (3 points)

Here is a line of code from a Ruby method:

```
line = (gets || return)
```

Imagining the reader to be a Java programmer with no knowledge whatsoever of Ruby, explain its operation in each of the possible cases that might arise when it executes. Ignore the open-command-line-arguments-as-files-and-read-them behavior of `gets`.

Problem 13: (8 points)

Write a Ruby iterator named `upto_limit(a, limit)`. The argument `a` is an array of integers; `limit` is an integer. `upto_limit` yields the values of `a` in turn (`a[0]`, `a[1]`, ...) continuing while the sum of the yielded values is less than or equal to `limit`. `upto_limit` returns `a`.

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

>> upto_limit([1,1,1,1],3) { |x| puts x }
1
1
1
=> [1, 1, 1, 1]

>> sum = 0
=> 0
>> upto_limit([10,20,30,40], 1000) { |x| sum += x }
=> [10, 20, 30, 40]
>> sum
=> 100

>> upto_limit([0,0,0,1], 0) { |x| puts x }
0
0
0
=> [0, 0, 0, 1]
```

Problem 14: (5 points)

Write a Ruby method `extract(s, m, n)` that extracts a portion of a string that represents a hierarchical data structure. m is a major index and n is a minor index. Major sections of the string are delimited by slashes and are composed of minor sections separated by colons. Here is a sample string:

```
/a:b/apple:orange/10:2:4/xyz/
```

It has four major sections which in turn have two, two, three and one minor sections. A call such as `extract(s, 3, 2)` should locate the third major section (`"10:2:4"` in the string above) and return the second minor section therein (`"2"`). If either section number is out of bounds, `extract` returns `nil`. Assume that `m` and `n` are greater than zero and that `s` is well-formed.

```
>> s = "/a:b/apple:orange/10:2:4/xyz/"
=> "/a:b/apple:orange/10:2:4/xyz/"

>> extract(s,1,1)    => "a"

>> extract(s,1,2)    => "b"

>> extract(s,3,3)    => "4"

>> extract(s,4,1)    => "xyz"

>> extract(s,4,2)    => nil

>> extract(s,10,1)   => nil
```

Hint: `Assume that "|20|1|300|".split("|")` produces `["20", "1", "300"]`.

Extra Credit Section (one point each unless otherwise indicated)

(1) Imagine that you have an ML function named `f` and a Ruby method named `f`. Does `[f]` produce an analogous result in both languages? If not, how do the results differ?

(2) For up to three points name three programming languages developed at the University of Arizona and give an example of a valid expression in each that involves an operator.

(3) For one point each, write `curry` and `uncurry` in ML.

(4) Assuming that `s` is a string, what is a Ruby expression that produces the same effect as `s.dup` but is both shorter and more difficult to type?

(5) What element of Ruby most closely corresponds to an anonymous function in ML?

(6) Simplify this Ruby expression: `if a[0] == nil then false else true end`

(7) Cite a contribution to knowledge made by Ralph Griswold.

(8) (Up to five points.) Offer some intelligent observations about the applicability of type deduction, or something similar, in Ruby.

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`

Last name, NetID

_____

CSC 372 Haskell Mid-term Exam
Feburary 28, 2014

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 and NetID in the box above.

This is a 45-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. We 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.

Feel free to use abbreviations, like "otw" for "otherwise".

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 five sheets.**

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

Problem 1: (15 points)

What is the type of the following values? If the expression is invalid, briefly state why.

Assume numbers have the type `Int`. Remember that the type of `True` is `Bool`.

```
"x"
```

```
(1,'x',[True])
```

```
["x":[]]
```

```
head
```

```
not . not . isLetter
```

```
map not
```

```
(+1) . (0<)
```

```
\x -> x + 1
```

```
[1,'2',"3"]
```

```
isDigit . chr . head . (:[]) . (+3) . ord . chr
```

Hint: the composition is valid. Here are some types:

```
ord :: Char -> Int
chr :: Int -> Char
isDigit :: Char -> Bool
```

Problem 2: (12 points) (two points each)

Using no functions other than helper functions you write yourself, implement the following Prelude functions.

Note: There will be a ½-point penalty for each case of not using the wildcard pattern (the underscore) where it would be appropriate.

`head`

`tail`

`length`

`sum`

`last` (Return `error "empty"` if the list is empty.)

`snd` (Returns second element of a 2-tuple)

Problem 3: (10 points)

Write a function `prswap` that swaps the elements of a list on a pair-wise basis. That is, the first and second elements are swapped, the third and fourth are swapped, etc.

ASSUME the list has an even number of elements but is possibly empty.

```
> prswap [1..6]
[2,1,4,3,6,5]

> prswap "abcd"
"badc"

> prswap [False,True,True,False]
[True,False,False,True]

> prswap []
[]

> :t prswap
prswap :: [a] -> [a]
```

There are **no restrictions** on this problem.

Problem 4: (10 points)

Write a function `rotabc` that changes `a`'s to `b`'s, `b`'s to `c`'s and `c`'s to `a`'s in a string. Only lowercase letters are affected.

```
> rotabc "abc"
"bca"

> rotabc "test"
"test"

> rotabc "aaaaa"
"bbbbb"

> rotabc "abababab"
"bcbcbc"

> rotabc "cabinet"
"abcinet"

> :t rotabc
rotabc :: [Char] -> [Char]
```

There are **no restrictions** on this problem but it must be written out in full detail—no ditto marks, abbreviations, etc. It must be ready for an ASU or UNC-CH CS graduate to type in.

This is essentially like `xlt "abc" "bca"` with `editstr` from assignment 1 but don't be tempted to use that!

Hint: My first "solution" for this one got a non-exhaustive match error on one of the tests.

Problem 5: (20 points) (ten points each)

For this problem you are to write **two separate versions** of a function named `bigTuples` (call it `bt`).

One version must not use any higher order functions (like assignment 1); the other version must not use any explicit recursion (like assignment 2, minus `warmup.hs`).

`bigTuples max tuples` produces a list of the 2-tuples in `tuples` whose sum is larger than `max`, i.e., for a tuple `(a,b)`, `a + b > max`.

```
> bigTuples 10 [(9,3),(3,4),(10,20)]
[(9,3),(10,20)]

> bigTuples 25 [(9,3),(3,4),(10,20)]
[(10,20)]

> bigTuples 100 [(9,3),(3,4),(10,20)]
[]

> bigTuples 1 []
[]

> take 5 $ bigTuples 1000 $ zip [1..] [1..]
[(501,501),(502,502),(503,503),(504,504),(505,505)]

> :t bigTuples
bigTuples :: (Num t, Ord t) => t -> [(t, t)] -> [(t, t)]
```

REMEMBER: You've got to write **two versions** of `bigTuples`!

Problem 6: (6 points) (5 for function, 1 for type)

Write a function `fml list` that returns a 3-tuple with the first, middle and last elements of a list.

```
> fml [1,2,3,4,5]
(1,3,5)

> fml [10]
(10,10,10)

> fml [1..101]
(1,51,101)

> fml "Haskell"
('H','k','l')
```

Assume the list is non-empty and has at least one element.

Restriction: You may not use the !! list indexing operator.

And, answer this question, too: What is the type of `fml`?

You didn't forget to state the type of `fml`, did you?!

Problem 7: (10 points)

Consider a function `separate s` that returns a 2-tuple with the digits and non-digits in the string `s` separated, with the initial order maintained:

```
> separate "July 4, 1776"
("41776","July , ")

> separate "Problem 7: (10 points)"
("710","Problem : ( points)")
```

Here is a partial implementation of `separate`, using `foldr`:

```
separate s = foldr f ([],[]) s
```

Your task on this problem is to write the folding function `f`. Remember that for `foldr`, the type of the folding function can be described as `elem -> acm -> elem`. Use `isDigit` to test for a digit.

Problem 8: (5 points)

Implement `map` in terms of a fold.

Your solution must look like this:

```
map f list = fold...
```

Note that the ellipsis starts right after the "`d`"—you'll need to decide which of `foldl1`, `foldr1`, `foldl`, or `foldr` you need to use!

It's ok to use an anonymous function but that is not required.

Problem 9: (2 points)

Without using explicit recursion, write `last` in point-free style. Hint: Tough, and easy to get backwards! Don't worry about handling empty lists.

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

(1) Who founded the University of Arizona Computer Science department and when? (2 points)

(2) Name two programming languages created **before 1990**.

(3) Name two programming languages created **after 1989**.

(4) Name one language feature you would expect to find in a language that supports imperative programming.

(5) whm often says "In Haskell we never change anything; we only make new things." What's an example of that?

(6) Things like cons lists, recursion, curried functions, and pattern matching on data structures are commonly used in functional programming but there's another capability that without which it's hard to do anything that resembles functional programming. What's that capability?

(7) What is relatively unique among programming languages about the way that Haskell handles strings and lists?

(8) What is a "partial application"?

(9) What is "syntactic sugar"?

(10) What's something we can represent with a tuple that we can't represent with a list?

Extra Credit Section (½ point each unless otherwise noted)

(1) What is the type of the composition operator?

(2) What's meant by "lexicographic comparison"?

(3) Name a programming language created at the University of Arizona.

(4) Haskell functions like `getLine` and `putStr` return an action. What does an action represent?

(5) What is the exact type of the list `[head, tail]`?

(6) If you only remember one thing about the Haskell segment of 372, what will it be? (Ok to be funny!)

(7) With Ruby in mind, cite an example of an imperative method and an example of an applicative method.

(8) Write a **Ruby** method `printN` that prints the numbers from 1 to N on a single line, separated by commas. (1 point)

Last name, NetID

_____

CSC 372 Ruby Mid-term Exam
April 11, 2014

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 and NetID in the box above.

This is a 45-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. We 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.

Feel free to use abbreviations.

Don't make a problem hard by assuming it needs to do more than is specifically mentioned in the write-up.

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 seven sheets.**

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

Problem 1: (21 points)

In this problem you are to write a Ruby program `tail.rb` that prints the last N lines of standard input, just like the UNIX `tail` command. Here's an example, after first demonstrating that the file `five` has five lines.

```
% cat five
one
two
three
four
five
% ruby tail.rb -2 < five
four
five
%
```

If the file has less than N lines, all lines are printed. Example:

```
% ruby tail.rb -10 < five       # prints all five lines
one
two
three
four
five
%
```

Exactly one command line argument should be specified and, with the leading dash, it should look like a negative integer, like "`-3`" or "`-100`". Otherwise, print "`Usage: tail -N`" and exit by calling `exit(1)`.

Examples of erroneous invocations:

```
% ruby tail.rb < five
Usage: tail -N

% ruby tail.rb 100 < five
Usage: tail -N
```

Behavior is undefined with the argument "`-0`". (In other words, ignore that case.)

Important: Make this simple by assuming you can hold the entire file in memory—don't imagine you need to use a circular queue or something like that!

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

(space for solution for `tail.rb`)

Quick reminder of operation:

```
% ruby tail.rb -4 < five
two
three
four
five
% ruby tail.rb 4 < five
Usage: tail -N
%
```

Problem 2: (17 points)

In this problem you are to write a method `load_facts(file_name)` that reads Prolog facts from the specified file and returns a Ruby hash that holds a representation of the facts. Here's a sample input file, `fcl.pl`, shown with `cat`:

```
% cat fcl.pl
food(apple).
%food(broccoli).
food(lettuce).

color(sky,blue).
color(dirt,brown).
color(grass,green).
color(x).

thing(apple,red,yes).
thing(a,b).
```

Usage, with the hash contents shown **formatted by hand**, to make it easier to read.

```
>> h = load_facts("fcl.pl")
=> {
"food"=>[["apple"], ["lettuce"]],
"color"=>[["sky", "blue"], ["dirt", "brown"],
          ["grass", "green"], ["x"]],
"thing"=>[["apple", "red", "yes"], ["a", "b"]]
}
```

Each of the functors `food`, `color`, and `thing` are keys in the hash. The value associated with each key is an array of arrays. Each entry in the inner arrays represents the term(s) for one fact. **Because there are two one-term facts for `food`, `h["food"]` references an array containing two one-element arrays.**

Similarly, `color` has three two-term facts and one one-term fact.

Empty lines or lines that start with a `%` are ignored. All other lines are well-formed Prolog facts with at least one term. Lines contain no whitespace.

To read from a file, use `f = File.open("x")` to open a file named `"x"` and then `f.gets` to read from it. `f.gets` returns `nil` at end of file. `f.close` closes it.

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

(space for solution for `load_facts`)

For reference, here are the facts and resulting hash again:

```
% cat fcl.pl
food(apple).
%food(broccoli).
food(lettuce).

color(sky,blue).
color(dirt,brown).
color(grass,green).
color(x).

thing(apple,red,yes).
thing(a,b).
```

```
>> h = load_facts("fcl.pl")
=> {
"food"=>[["apple"], ["lettuce"]],
"color"=>[["sky", "blue"], ["dirt", "brown"],
          ["grass", "green"], ["x"]],
"thing"=>[["apple", "red", "yes"], ["a", "b"]]
}
```

Problem 3: (15 points)

This problem is a partner to `load_facts`. You are to write a Ruby method `print_preds` that takes the hash produced by `load_facts` (the previous problem) and prints a list of predicate indicators.

Usage, using the facts from the previous problem:

```
>> h = load_facts("fcl.pl")
=> {
"food"=>[["apple"], ["lettuce"]],
"color"=>[["sky", "blue"], ["dirt", "brown"],
          ["grass", "green"], ["x"]],
"thing"=>[["apple", "red", "yes"], ["a", "b"]]
}

>> print_preds(h)
color/1
color/2
food/1
thing/2
thing/3
=> nil
```

Predicates are shown in alphabetical order, with a predicate indicator (*`functor/N`*) for each of the forms present. We can see that the hash `h` holds one-term facts for `food`, one- and two-term facts for `color`, and two- and three-term facts for `thing`.

Here are some possibly useful things:

`Hash#keys` returns an array of the keys in a hash.
`Array#sort` returns a copy of the array with the entries sorted.
`Array#min` and `Array#max` return the minimum and maximum values in an array.
`Enumerable#map` is the iterator analog for Haskell's `map` function.

Problem 4: (5 points)

Write a method `multstring(spec, s)` that behaves like this:

```
>> multstring("3,1,5", "x")
=> "xxx,x,xxxxx"

>> multstring("3,0,2,0", "Abc")
=> "AbcAbcAbc,,AbcAbc,"

>> multstring("10", "")
=> ""
```

The first argument, `spec`, is a comma-separated list of non-negative integers. The result is a string that consists of comma-separated replications of the second argument (`s`) corresponding to each integer in turn. Assume that `spec` is well-formed and contains no whitespace.

Problem 5: (4 points)

Write a method `addrev` such that after calling `addrev`, the unary minus operator applicatively reverses a string:

```
>> addrev
=> nil

>> x = -"testing"
=> "gnitset"

>> -x
=> "testing"

>> x
=> "gnitset"
```

Problem 6: (11 points) One point each unless otherwise indicated

(a) Write a regular expression that is equivalent to `/x+/` but that doesn't use the `+` operator.

(b) Write a regular expression that is equivlent to `/ax|bx|cx/` but that doesn't use the `|` operator.

(c) Describe in English the strings matched by `/a*b+c$/`

(d) (3 points) Write a regular expression that matches only strings that are binary constants like these:

```
"0b0"
"0B11111"
"0b01010101"
```

The first two characters are a "`0`" and an upper- or lower-case "`B`". One or more "`1`"s or "`0`"s follow.

Here are two non-matches: "`00b1`" (extra leading zero), "`0b12`" (trailing non-0/1).

(e) (5 points) Write a regular expression that matches a string if and only if it is a comma-separated sequence of integers **greater than or equal to 10**. Examples of matches:

```
"10, 11, 12"
"30,200,500"
"100"
```

One optional space may appear after each comma but that is the only place a space may appear. **Leading zeros are not permitted!**

Here are three non-matches: "`10, 020`" (has a leading zero), "`7,11`" (7 is less than 10), "`1x`" (has a trailing "`x`").

Problem 7: (7 points)

Write an iterator `take_while(a)` that calls its block with each element of the array `a` in turn. As long as the block returns true, array elements are accumulated. When the block first returns a non-true value or when the array elements are exhausted, an array of the accumulated elements is returned.

```
>> take_while([1,2,3,4,5]) { true }
=> [1, 2, 3, 4, 5]

>> take_while([1,2,3,4,5]) { false }
=> []

>> take_while([1,2,3,4,5]) { |e| e.odd? }
=> [1]

>> take_while([1,2,3,4,5,3,2]) { |e| e < 5 }
=> [1, 2, 3, 4]

>> take_while("eat peas all day".split) { |w| w =~ /e/ }
=> ["eat", "peas"]
```

Note: This is much like `Enumerable#take_while`, so you can't use that method in your solution! (It's like Haskell's `takeWhile`, too.)

Problem 8: (12 points)

(a) Who invented Ruby? (1 point)

(b) Consider a Ruby method `last(x)` that returns the last element of a string or array. Why would `last` be awkward to write and use in Java? (2 points)

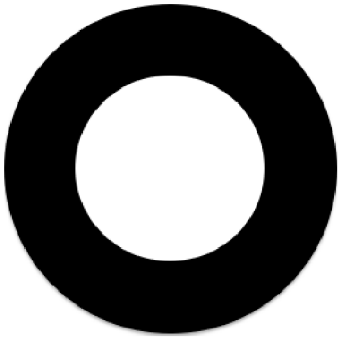
(c) What's the effect of adding "`include Enumerable`" to a class definition? (2 points)

(d) Cite one way in which Ruby's `if-then-else` is like Haskell's `if-then-else` and one way in which it is different. (2 points)

(e) A Java newcomer to Ruby might think that `private` and `attr_reader` are keywords but they aren't. What are they? (2 points)

(f) What's the essential difference between dynamic typing and static typing? (3 points)
Hint: If your answer contains `/(interprete|compile)[rd]?/` there's a fair chance it'll be wrong.

Problem 9: (8 points)



Recall the `Shape`/`Circle`/`Rectangle` inheritance hierarchy from the slides. `Shape`'s constructor requires a single argument: a string that serves as a label for the shape. `Shape#label` returns the label.

In this problem you are to create a subclass of `Shape` named `Ring`. An instance of `Ring` represents the region between two concentric circles (the black area in the drawing to the right). Along with a label that is passed to `Shape`'s constructor, `Ring`'s constructor takes the radii of the two circles but they may specified be in either order. If the two circles have the same radius, the area of the ring is zero.

Your implementation of `Ring` should have five methods: a constructor, `area`, `inner`, `outer`, and `inspect`. The methods `area`, `inner`, and `outer` return the area of the ring, the inner radius and the outer radius, respectively. `inspect` displays the inner (smaller) radius followed by the outer radius.

Assume both radii are non-negative. `Math::PI` is π.

Examples:

```
>> r1 = Ring.new("a",3.2,1.9)
=> Ring a (1.9-3.2)       # This line displays the result of inspect. The inner
                          # radius, 1.9, is shown first.

>> r1.area
=> 20.82875929330033

>> r1.inner
=> 1.9

>> r1.outer
=> 3.2

>> Ring.new("x",3,3).area
=> 0.0
```

Write your solution to the right or on the next page.

(space for solution for `Ring`)

Examples (repeated):

```
>> r1 = Ring.new("a",3.2,1.9)
=> Ring a (1.9-3.2)        # This line displays the result of inspect. The inner
                           # radius, 1.9, is shown first.

>> r1.area
=> 20.82875929330033

>> r1.inner
=> 1.9

>> r1.outer
=> 3.2

>> Ring.new("x",3,3).area
=> 0.0
```

Extra Credit Section (½ point each unless otherwise noted)

(a) What does whm consider to be the all-time greatest Original Thought ever put forth by one of his 372 students?

(b) Based on the Prolog we've covered, is Prolog statically typed or dynamically typed? (And why?)

(c) What thing in Ruby is closest to a Prolog atom?

(d) Which is the oldest of Java, Haskell, and Ruby?

(e) The term "mixin" came from a business that sold what to college students?

(f) Regular expressions are Type _____ languages in the _____ hierarchy of languages.

(g) Predict the median score on this exam. (The median is the "middle" value in a range of values.)

(h) If you only remember one thing about Ruby, what will it be? (Ok to be funny!)

(i) What's the stupidest thing in Ruby? Can't be the same as (h) or (j). (1 point)

(j) What's the best thing in Ruby? Can't be the same as (h) or (i). (1 point)

(k) Implement `attr_reader`. (1 point)

Last name, NetID

_____

CSC 372 Final Exam
Wednesday, April 14, 2014

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 and NetID 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, aside from the sheet mentioned on Piazza.

If you have a question, raise your hand. We 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.

Feel free to use abbreviations.

Don't make a problem hard by assuming it needs to do more than is specifically stated in the write-up.

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 seven sheets.**

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

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.

Note: If you were at last night's review session you can't cite the Python feature that was mentioned. (Please ask if you were at the review session and have a question about this restriction!)

Problem 2: (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.

Restriction: No library predicates may be used other than `is/2`.

`last(?List,?Elem)` specifies the relationship that `Elem` is the last element of `List`, which is assumed to be non-empty.

`member(?Elem, ?List)` specifies the relationship that `Elem` is a member of `List`.

`length(?List, ?Len)` specifies the relationship that `List` is `Len` elements in length.

Problem 3: (8 points)

Write a Prolog predicate `insrel(+List, -WithRels)` that instantiates `WithRels` to a copy of `List` with one of the atoms <, >, and = inserted between each pair of values to reflect the relationship between those values. Assume all values are numbers. Examples:

```
?- insrel([5,3,7,7,0],L).
L = [5, >, 3, <, 7, =, 7, >, 0] .

?- insrel([10,10],L).
L = [10, =, 10] .

?- insrel([7],L).
L = [7] .

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

Only the first result is of interest; don't worry about behavior if the user responds with a semicolon.

Note that <, >, and = are valid symbolic atoms and are therefore shown without quotes.

Hint: Write a helper predicate `which(+X,+Y,-Rel)` that works like this:

```
?- which(3,4,Op).
Op = < .
```

Problem 4: (7 points)

Write a Prolog predicate `alltails(+List, -T)` that first instantiates `T` to the tail of `List`. If an alternative is requested, it generates the tail of the tail of `List`, and so forth, thus generating "all the tails".

Restriction: Only "cons", unification, and recursive calls to `alltails` may be used. In particular, you can't use `append` or write your own version of `append` and use it.

```
?- alltails([10,20,30,40],T).
T = [20, 30, 40] ;
T = [30, 40] ;
T = [40] ;
T = [] ;
false.

?- alltails([x],T).
T = [] ;
false.

?- alltails([],T).
false.
```

Don't forget the restriction!

Problem 5: (10 points)

Write a Prolog predicate `pinch(+List, -Pair)` that instantiates `Pair` to a series of `pair/2` structures. The first `pair` structure has the first and last values of `List`. The second `pair` has the second and next-to-last values of `List`. And so forth. Example:

```
?- pinch([a,b,c,d,e,f],P).
P = pair(a, f) ;
P = pair(b, e) ;
P = pair(c, d) ;
false.
```

If a list has an odd number of values, the last `pair` has two copies of the middle element.

```
?- pinch([1,2,3],P).
P = pair(1, 3) ;
P = pair(2, 2) ;
false.

?- pinch([1],P).
P = pair(1, 1) ;
false.
```

`pinch` fails on an empty list.

```
?- pinch([],P).
false.
```

Restriction: You may not use any arithmetic in your solution.

You may assume the presence of `last(?List,?Elem)`.

Problem 6: (5 points)

Write a Prolog predicate `sumvals/0` that consolidates all `v/1` facts into a single fact that contains the sum of the terms of those facts. Assume all terms are numbers. There may be any number of `v/1` facts.

```
?- v(X).
X = 1 ;
X = 7 ;
X = 2.

?- sumvals.
true.

?- v(X).
X = 10.

?- sumvals.
true.

?- v(X).
X = 10.
```

If no `v/1` facts exist, `sumvals` creates the fact `v(0).`

```
?- v(X).
false.

?- sumvals.
true.

?- v(X).
X = 0.
```

Problem 7: (12 points)

This is the problem you were told to expect that is like `connect` from assignment 8.

`path(+Start, +End, +Moves, -Path)` instantiates `Path` to a series of values from `Moves` that describe a path from `Start` to `End`, two points on a Cartesian plane. All alternatives are produced if requested. Each move can be used only once in a given path. Example:

```
?- path(p(0,0), p(3,4), [m(4,4),m(3,3),m(0,1),m(-1,0)], Moves).
Moves = [m(4, 4), m(-1, 0)] ;
Moves = [m(3, 3), m(0, 1)] ;
Moves = [m(0, 1), m(3, 3)] ;
Moves = [m(-1, 0), m(4, 4)] ;
false.
```

The call asks for a path from `(0,0)` to `(3,4)` using some combination of movements, which are `m/2` structures that specify relative changes in X and Y, respectively. Here's the set of moves from above:

```
m(4,4)    Move right 4 and up 4.
m(3,3)    Move right 3 and up 3.
m(0,1)    Move up 1.
m(-1,0)   Move left 1.
```

In the following case there's both an empty solution and a solution that uses all three moves.

```
?- path(p(10,20), p(10,20), [m(-2,0),m(1,2),m(1,-2)], Moves).
Moves = [] ;
Moves = [m(-2, 0), m(1, 2), m(1, -2)] ;
Moves = [m(-2, 0), m(1, -2), m(1, 2)] ;
...four additional permutations aren't shown...
```

If no suitable sequence of moves exists, `path` fails.

```
?- path(p(0,0), p(1,1), [m(1,0),m(0,2),m(1,0)], Moves).
false.
```

As the preceding example demonstrates, a successful series of movements must terminate at the destination, not merely pass through it.

Problem 8: (10 points)

Using the parsing method supported by Prolog's grammar rule notation write a predicate `ptime(+Spec,-Mins)` that parses atoms that represent time durations, like `'10m'`, `'5h'` and `'10:20'`, and instantiates `Mins` to the number of minutes represented by `Spec`.

Durations will either be a number (an integer >= 0) followed by "`m`" or "`h`", or two numbers separated by a colon. `'10m'` is ten minutes; `'2h'` is two hours—120 minutes. `'2:30'` is two hours and thirty minutes—150 minutes. Something like `'1:2000'` is valid, too (2,060 minutes).

```
?- ptime('10m',Mins).
Mins = 10.

?- ptime('2h',Mins).
Mins = 120.

?- ptime('2:30',Mins).
Mins = 150.

?- ptime('2:3',Mins).  % not required to be '2:03', for example
Mins = 123.
```

Assume you have a grammar rule `int(N) --> ...` that recognizes a non-negative integer and instantiates `N` to the recognized value. Example:

```
?- int(I,['2','3'],[]).
I = 23 .

?- int(I,['2','3',x],Left).
I = 2,
Left = ['3', x] ;
I = 23,
Left = [x] ;
false.
```

Problem 9: (7 points)

Write a **Haskell** function `ckconn` that takes a list similar to that used by a8's `connect.pl` and returns `True` or `False`, depending on whether the exact sequence and orientation of cables represents a valid connection.

Example:

```
> ckconn 'm' [('f',10,'m'), ('f',7,'m')] 'f'
True
```

In contrast to the Prolog version, the list of cables appears between the left and right endpoints. `ckconn` does not check the length of the configuration. Note that cables are specified with three-tuples, not lists.

Note that `ckconn` checks the list as-is. It doesn't consider alternatives in any way. For example, simply flipping the last cable in the configuration above produces `False`:

```
> ckconn 'm' [('f',10,'m'), ('m',7,'f')] 'f'
False
```

Any number of cables may be specified. If no cables are specified, `ckconn` returns `False`.

```
> ckconn 'm' [] 'f'
False
```

Your solution may be recursive or not; your choice!

Important: Include a type declaration for `ckconn`. I'll get you started:

```
ckconn :: Char ->
```

Problem 10: (7 points)

Write a **Haskell** function `connlen` that takes a list of the same type as `ckconn` and **prints** either the length of the configuration, like "`25 feet`" or "`nope`", if the configuration is not valid.

```
> connlen 'm' [('f',10,'m'), ('f',7,'m')] 'f'
17 feet

> connlen 'm' [('f',10,'m'), ('f',7,'m')] 'm'
nope
```

Restriction: Like on assignment 2, your solution for `connlen` must be non-recursive.

You may use assume a working `ckconn` from the previous problem and use it in `connlen`.

Remember: `connlen` produces output; its final result has type `IO ()`. (Not `String`, for example.)

Problem 11: (14 points)

Write a **Ruby program** `shuffile.rb` that reads lines from standard input, shuffles them, and then writes them to standard output.

`shuffile` requires one command line argument, `-N`, which specifies that lines are to be handled in blocks of N lines. Assume that standard input consists of a multiple of N lines.

The UNIX utility `seq` can be used to output a sequence of numbers. We'll use it to demonstrate `shuffile`'s behavior. `seq 6` outputs the first six integers:

```
% seq 6
1
2
3
4
5
6
```

With the argument `-1` the input lines are shuffled just like you might shuffle a deck of cards:

```
% seq 6 | ruby shuffile.rb -1
5
6
3
1
4
2
```

With the argument `-2` the input lines are treated as pairs—the first and second lines are kept together, as are the third and fourth, and the fifth and sixth. With the output of `seq 6` as input, the line "3" will always be followed by the line "4", for example.

```
% seq 6 | ruby shuffile.rb -2
3
4
1
2
5
6
```

There are only two possible results of `seq 6 | ruby shuffile.rb -3`. Here's one of them:

```
% seq 6 | ruby shuffile.rb -3
4
5
6
1
2
3
```

There must be exactly one command line argument and it must be of the form `-N`. If not, "`Oops!`" is printed and `exit(1)` is called:

```
% seq 6 | ruby shuffile.rb
Oops!

% seq 6 | ruby shuffile.rb 3
Oops!

% seq 6 | ruby shuffile.rb -2 3
Oops!
```

All the examples above use a six line input but `shuffile.rb` should be able to handle any number of lines of any length with arbitrary content, not just numbers.

Important: Note that there is an `Array.shuffle` method:

```
>> ["please","shuffle","me","up"].shuffle
=> ["up", "please", "shuffle", "me"]
```

Problem 12: (8 points, one point each)

(a) Who founded UA's CS department, and in what year?

(b) Why are Prolog warnings about singleton variables worth paying attention to?

(c) What's the fundamental difference between predicates like `member/2` and `append/3` in Prolog and their superficial analogs in other languages?

(d) Draw the box for Prolog's four-port model and label the ports.

(e) "cons" lists are found in many languages that make use of recursion. What is it about cons lists that makes them well-suited for recursive functions?

(f) Consider this claim: Prolog's grammar rule notation, like `sentence --> article, noun, verb`, is an example of syntactic sugar. Present an argument that either supports that claim or refutes it.

(g) Write a simple **Haskell function** and show an example of using a partial application of that function.

(h) What's a very good reason that `ckconn` above uses tuples instead of lists to represent the cables?

Extra Credit Section (½ point each unless otherwise noted)

(a) What movie inspired the "Original Thought" bonus?

(b) whm came to graduate school here at UA specifically to join a research team developing a programming language. What was the language?

(c) To what current CS faculty member does whm attribute the following quotation?
"When you come to a problem you can lean forward and type, or you can sit back and think."

(d) The Prolog 1000 is a compilation of applications written in Prolog and related languages. What's a little odd about it?

(e) Jean Ichbiah, Ada's designer, was said to have once predicted that in ten years only two programming languages would remain in use. Ada was one of the languages. What was the other?

(f) Cite a significant contribution to computer science made by Grady Booch.

(g) What's a language that has/had an "arithmetic if", one form of which looks like this:

```
IF (I-J) 100, 110, 120
```

(h) In terms of creators, what do `vim` and Java have in common?

(i) whm would like to recycle a8's `buy.pl` in a future semester but needs mores `dontmix` facts with an element of humor. What's another pair he could use?

(j) How many students earned the "close reading" bonus on assignment 8?

Last name

_____

CSC 372 Mid-term Exam
Thursday, March 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 60-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.

Feel free to use abbreviations, like "otw" for "otherwise", "lt" for "longerThan". **Use `S` as a synonym for `[Char]`. Use `I` for `Int`**.

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 seven sheets.**

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

Problem 1: (15 points)

What is the type of the following values? If the expression is invalid, briefly state why.

Assume numbers have the type `Int`. Remember that the type of `True` is `Bool`.

```
'x'
```

```
("len", "", "")
```

```
(:)
```

```
[(1,'a'),('b',2)]
```

```
map length
```

```
(+1) . (0<)
```

```
head
```

```
(True, (False, "x"))
```

```
length . (\x -> [(x,x)]) . head . init . tail
```

Hint: the composition is valid.

```
[1..] < [2..]
```
(Remember: I want the type!)

Problem 2: (12 points) (two points each)

This problem is like `warmup.hs`—write the following Haskell Prelude functions.

Each instance of poor style or needlessly using other Prelude or helper functions will result in a deduction.

Remember this order of preference for handling cases: patterns, guards, if-else. Be sure to use the wildcard pattern (underscore) when appropriate.

IGNORE empty list cases for `head`, `tail`, and `maximum`. There's no need to specify function types.

`head`

`tail`

`length`

`max` (Hint: `max 2 3` is `3`)

`maximum` (`maximum [5,2,3]` is `5`. Ok to abbreviate as `mm`.)

`filter` (`filter odd [1,2,3]` is `[1,3]`. Ok to abbreviate as `flt`.)

DID YOU REMEMBER TO USE WILDCARDS WHEN APPROPRIATE?

Problem 3: (14 points) (7 points each)

For this problem you are to **write both recursive and non-recursive versions** of a function `tupruns` of type `[(Int, a)] -> [a]` that behaves like this:

```
> tupruns [(3, 'a'), (2, 'b'), (4, 'c')]
"aaabbcccc"

> tupruns [(1,'#'),(0,'$')]
"#"

> tupruns  (zip [1..8] "abcdefgh")
"abbcccddddeeeeeffffffgggggggghhhhhhhh"
```

Just like on assignment 2, the recursive version must not use any higher-order functions. Just like on assignment 3, write the non-recursive version imagining that you just don't know to how to write a recursive function.

For one bonus point each on the non-recursive version:
(1) Use point-free style.
(2) Do not use any helper functions or anonymous functions.

`concat` and `replicate` may be useful:

```
> concat ["a", "...", "c"]
"a...c"

> replicate 3 5
[5,5,5]
```

Problem 4: (3 points)

Write a function `flattups` that "flattens" a list of 3-tuples into a list of the values in those tuples.

```
> :t flattups
f :: [(a, a, a)] -> [a]

> flattups [(10,20,30),(3,4,5),(1,2,3)]
[10,20,30,3,4,5,1,2,3]


> flattups [('H','i','!')]
"Hi!"
```

There are no restrictions on this problem. You may call it just "`ft`".

Problem 5: (7 points)

The following function removes consecutive duplicates from a list:

```
dropConsecDups list = foldr ff [] list
```

Usage:

```
> dropConsecDups [3,3,1,5,7,5,5,7,7,7]
[3,1,5,7,5,7]

> dropConsecDups "a looooooooooooonnnnnnnnnng one!"
"a long one!"
```

For this problem you are to write a folding function `ff` that will work with `dropConsecDups` as shown above.

Problem 6: (6 points)

Write a Haskell program that reads a file named on the command line and prints the mean (average) length of the lines in the file.

Usage:

```
$ cat avglen.1
xxx
y
zz

$ runghc avglen.hs avglen.1
2.0
```

Note the computation: *(3+1+2)/3 = 2 characters per line*

DO NOT WORRY about integer/float division issues. **Assume** that `3/2` produces `1.5`!

Assume there's at least one line in the file.

Be sure that a newline follows the value that's output. Recall that `show 3.2` is `"3.2"`. Here's a main program, similar to the ones supplied for `group.hs` and `avg.hs`. Your job is to write `avglen`.

```
main =
    do
       args <- getArgs
       bytes <- readFile (head args)
       putStr (avglen bytes)
```

Problem 7: (7 points)

Now it's time for some Ruby problems.

Write a Ruby version of the Haskell program in the previous problem. The Ruby version reads from standard input rather than opening a file specified on the command line:

```
$ cat avglen.1
xxx
y
zz

$ ruby avglen.rb < avglen.1
2.0
```

Use `printf("%.1f\n", ...)` to produce the final output.

Unlike the Haskell version this Ruby version must properly handle the math to get a `Float` result, not a truncated value like `3/2 == 1`.

Problem 8: (7 points)

Write a Ruby program `nwords.rb` that reads lines from standard input and writes the first N words on each line to standard output.

N is specified by a `-N` command line argument that is assumed to be present. If N is larger than the number of words on a line, all the words on that line are output.

```
$ cat nwords.1
a few words to
test the
operation
of nwords.rb on this exam.

$ ruby nwords.rb -3 < nwords.1
a few words
test the
operation
of nwords.rb on

$ cat nwords.2
aa bb cc dd ee ff gg
hh ii jj
kk
ll mm nn oo pp

$ ruby nwords.rb -2 < nwords.2
aa bb
hh ii
kk
ll mm

$ ruby nwords.rb -100 < nwords.2
aa bb cc dd ee ff gg
hh ii jj
kk
ll mm nn oo pp
```

Note the following behavior for *array*`[n, m]`:

```
>> w = "a b c".split
=> ["a", "b", "c"]

>> w[0,2]
=> ["a", "b"]

>> w[0,10]
=> ["a", "b", "c"]
```

SPACE FOR SOLUTION ON NEXT PAGE

(space for solution for `nwords.rb`)

For reference:

```
$ cat nwords.2
aa bb cc dd ee ff gg
hh ii jj
kk
ll mm nn oo pp

$ ruby nwords.rb -2 < nwords.2
aa bb
hh ii
kk
ll mm

$ irb
>> w = "a b c".split     => ["a", "b", "c"]

>> w[0,2]                => ["a", "b"]

>> w[0,10]               => ["a", "b", "c"]
```

Problem 9: (6 points)

Write Ruby method `chunkstr(s, n)` that returns an array of consecutive `n`-long substrings of the string `s`. Partial substrings are not included. Assume `n > 0`. `chunkstr` does not change `s`! (Maybe use `String#dup`.)

```
>> chunkstr("abcdef",3)
=> ["abc", "def"]

>> chunkstr("abcdef",1)
=> ["a", "b", "c", "d", "e", "f"]

>> chunkstr("abcdef",5)
=> ["abcde"]
```

Problem 10: (6 points) (one point each unless otherwise indicated)

The following questions and problems are related to Haskell.

(1) Add parentheses to the following type to explicitly show the associativity of the `->` operator.

```
(Int -> Int) -> Int -> Int
```

(2) Fully parenthesize the following expression.

```
f   g   a   +   x   y   z  +  f1  (a, b)  c
```

(3) Rewrite the following expression to use as few parentheses as possible.

```
len ((map f) (head (lines bs)))
```

(4) Describe in English the data structure matched by this pattern: `[t:h]`

(5) Consider the following definitions for a function that tests whether a list is empty. Each is shown with the types that Haskell infers for them:

```
empty1 :: [t] -> Bool
empty1 [] = True
empty1 _  = False

empty2 :: Eq a => [a] -> Bool
empty2 x
    | x == [] = True
    | otherwise = False
```

What's causing the type of the functions to differ? What's an example of a list that would work with `empty1` but not `empty2`? (Two points.)

Problem 11: (7 points) (one point each unless otherwise indicated)

The following questions and problems are related to Ruby.

(1) Write a Ruby iterator `itr` that behaves like this:

```
>> itr(3) {|x| puts x}
6
```

(2) In the Ruby code `$x = N * 5`, we know that `$x` is a _____ and `N` is a _____.

(3) Given `h = {}`, write an expression such that after it is evaluated, `h["x"]` is `10`.

(4) What's a big mistake in the following statement?
"The `if-else`, `while`, and `for` statements are examples of Ruby control structures."

(5) A method named `show_match` is used extensively on the regular expression slides. Briefly, what does it do?

(6) What are the minimum and maximum lengths of strings that can be matched by the following regular expression? (Be careful!)

```
[Aa]..[Zz]0x9?
```

(7) Write a Ruby method that exemplifies duck typing. There's no need for any explanation!

Problem 12: (10 points) (one point each unless otherwise indicated)

Answer the following general questions. Keep your answers brief for questions—assume that the reader is a 372 classmate who just needs a quick reminder.

(1) Who founded the University of Arizona Computer Science department and when?

(2) Name a language that was created before Ruby. Name a language that was created after Ruby.

(3) whm believes the acronym LHtLaL appears nowhere on the web except in his slides. What does it stand for?

(4) What are two aspects of a "paradigm", as described in Kuhn's *The Structure of Scientific Revolutions*? Here are two **wrong** answers: functional and object-oriented.

(5) Perhaps the most fundamental characteristic of a functional programming language is that it permits higher-order functions to be written. What's the language feature that's required in order to write a higher-order function?

(6) What's an important difference between an imperative method and an applicative method? (Hint: it doesn't concern the method name!)

(7) whm often says "In Haskell we never change anything; we only make new things." What's an example of that?

(8) In general, every expression in a programming language has three aspects. What are those three? Hint: one of them starts with a "v". (1.5 points)

(9) Briefly define the term "programming language". (1.5 points)

Extra Credit Section (½ point each unless otherwise noted)

(1) Predict the median score on this test. (The median is the "middle" value in a range of values.)

(2) Add a dunsel to the following function definition

```
f x = take 3 x ++ drop 3 x
```

(3) Why was \ chosen for use in Haskell's lamba abstraction syntax?

(4) To whom does whm attribute the following quote?
"When you hit a problem you can lean forward and type or sit back and think."

(5) What's the basic idea of whm's so-called "*O(1)* navigation"?

(6) What is the exact type of the list `[head, tail]`?

(7) What's interesting about the ASCII code sequence from 60 through 62?

(8) What Ralph say when a young and eager graduate student put forth a list of potential new features for Icon?

(9) If you only remember one thing about the Haskell segment of 372, what will it be? (Ok to be funny!)

(10) What is Matz' full name?

(11) Write a good extra credit question and answer it.

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 one-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:

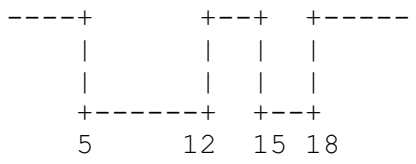
```
?- inout([1,2,3,4,5],R).
R = 3 ;
R = 4 ;
R = 2 ;
R = 5 ;
R = 1.

?- 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:

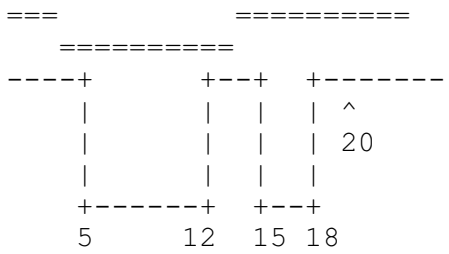


Pits are represented with `pit/2` facts, which have a starting position and a width:

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

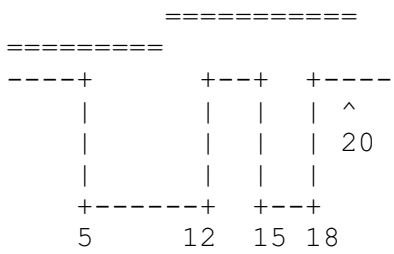
There may be any number of `pit/2` facts. Pits never overlap. Pits always have ground between them.

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.



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.

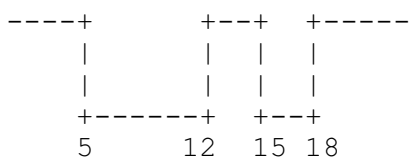
Here's an example of an invalid crossing, with the sequence `[9, 11]`. It's invalid because the two planks meet over a pit, at position 10. `[11,9]` and `[16,9]` would be invalid, too.



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)`:



The predicate you are to write is `cross(+Planks, +Distance, -Solution)`. 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.

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

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

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

The following questions and problems are related to Prolog.

(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 one word 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?