CSc 372, Spring 1996
Mid-term Examination Solution Key

Problem 1 (6 points):

State a definition for the term "programming language".

In the slides, a programming language is said to be "a notation for the description of computation".

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

Ideal answers included "a looping construct" and "the ability to change values via assignment".

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

Ideal answers included "recursion", "functions", and "use of functions as values".

Problem 2 (4 points):

Write ML expressions having the following types:

```
int * string list

     (1, ["x"])

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

     (1, (1, [(1,1)]))
```

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

     fun g(1) = 1
     fun f(g, 1) = g(1) = 1

g: int -> int -> int list

     fun f 1 2 = [1, 2]
```

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?   string * int list * (string -> int) -> bool

g?   string -> int

h?   int list -> bool

a?   string

b?   int list

x?   int
```

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.

```
exception NotOne;
fun f(1) = 1
 |  f(n) = raise NotOne
```

Problem 5 (3 points):

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

There is a type-compatible tuple that is not matched by any of the patterns defining a function.

What is one possible implication of the warning?

A match exception may be encountered when the function is executed.

Write an ML function that would produce that warning.

```
fun f(1) = 1
```

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?   [[1]]
```

```
y?   (2, [3])
```

```
z?   [4, 5, 6]
```

```
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!)

```
fun (x,y,z::xs) = 21
```

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

```
'a * 'b * 'c list -> int
```

Problem 7 (5 points):

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

```
fun length([]) = 0
 |  length((_:int)::xs) = 1 + length(xs);
```

Many persons defined a length function with type `'a list -> int`. Others, in trying to force the `int list` type ended up with a function that summed the elements in the list instead of counting them.

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`.

```
fun avgLen(L) =
     let
          fun sum([]) = 0
           |  sum(x::xs) = x + sum(xs)
          val lens = map length L
     in
          if length(L) = 0 then raise EmptyList
            else real(sum(lens)) / real(length(L))
```

This problem was not intended to be difficult but in fact it had a relatively low success rate. The key observation is that the essence of the problem is calculate the average of a list of integers.

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`.

```
fun F_L_eq([]) = false
 |  F_L_eq([x]) = true
 |  F_L_eq(x::xs) =
        let
            fun last([x]) = x
             |  last(_::xs) = last(xs)
        in
            x = last(xs)
        end
```

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`.

```
fun f(FL, []) = []
 |  f(FL, v::vs) =
     let
          fun apply_each([], x) = []
           |  apply_each(f::fs, x) = f(x)::apply_each(fs, x)
     in
          apply_each(FL, v)::f(FL, vs)
```

```
end
```

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?

Lists in ML must be homogeneous—all elements must be of the same type. Elements in an Icon list can be of differing types.

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

```
[1, 1.0]
```

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

     fun f(g::gs) = g("x") + 1;

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

     fun g "" 1 2.0 = [true]
```

Another way:

```
     fun g s i r = [s = "x", i = 1, r = 1.0];
```

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`.

```
     procedure Reverse(s)
             r := ""
             every r := !s || r
             return r
     end
```

Another way:

```
     procedure Reverse(s)
             r := ""
```

```
        every i := 1 to *s do
                r := s[i] || r
        return r
end
```

I had intended `Reverse` to be a trivial problem to serve as a warm-up for the Icon portion of the exam, but only a handful of persons produced a solution that would actually work.

A number of persons produced solutions that used list manipulation functions such as `push` and `pull` to manipulate strings, but in fact those functions don't work on strings. Despite that, such solutions typically received full credit.

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.

```
procedure Point(s)
    s ? {
            ="(" &
            x := tab(many(&digits)) &
            "," &
            y := tab(many(&digits)) &
            =")" & pos(0) &
            return [integer(x),integer(y)]
        }

end
```

A number of persons tried to solve this with `split`, but if `split` was used you needed to use the three-argument form and examine each piece produced. Most of the split-based solutions would accept strings such as `",,2,,,3,,"` or `"()()2()3()"`, or worse.

Problem 15 (8 points)

Write an Icon program that reads standard input and produces a histogram of line lengths encountered.

```
procedure main()
    hist := table("")

    while line := read() do
        hist[*line] ||:= "*"

    every pair := !sort(hist) do
        write(pair[1], "\t", pair[2])

end
```

Problem 16 (9 points):

Imagine a file with a format such as this:

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

...

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.

```
procedure main(args)
    len := args[1] | 10

    lines := []

    while line := read() do
        put(lines, line)

    lines := sort(lines)
    out := ""
    every line := !lines do {
        out ||:= (line ? ( tab(upto('.')+1) & tab(0) ))
        }

    out ? {
        while write(move(len))
        write(tab(0))
        }
end
```

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.

```
procedure cons(x,y)
     return [x] ||| y
end
```

It was also satisfactory to use a solution such as this:

```
procedure cons(x,y)
     return push(y,x)
end
```

if it was noted that the approximation was poor because list `y` is changed as a side-effect.

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.

```
procedure size(x)
    count := 0
    every !x & count +:= 1
    return count
end
```

Problem 19 (6 points) (*):

Write an Icon program to read standard input and print out the largest integer found in the input. ...

```
procedure main()
    while line := read() do {
        line ? while tab(upto(&digits)) do {
                val := tab(many(&digits))
                if /maxval | (val > maxval) then
                    maxval := val
                }

        }

    write(\maxval|"No integers")
end
```

Several persons approached this problem with `split`, but it is necessary to split on `~&digits` rather than whitespace to achieve a correct solution.

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.

An argument can be made either way and therefore, answering either "yes" or "no" earned a point. A reasonable argument of any sort earned two more points.

I would argue that most of the functional solutions we studied could be implemented in Icon with a minimum of difficulty, aside from Icon lacking an equivalent to ML's pattern matching facility. (It is said that Icon supports pattern matching via string scanning, but that is a completely different facility that shares the same name.)

On the other hand, Icon has nothing like anonymous functions, composition, or currying and therefore Icon's handling of functions as values, a cornerstone of functional programming, falls short.

CSc 372, Spring 1996
Final Examination Solution Key

Problem 1 (10 points):

What is the difference between a class and an object?

> An object is an instance of a class. In software, classes are used to create objects, specifying structure and behavior. In the real-world, classes are used to group objects.

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

> Data members exist solely to support the operation of the function members. A class with no function members should have no data members.

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

> Never.

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

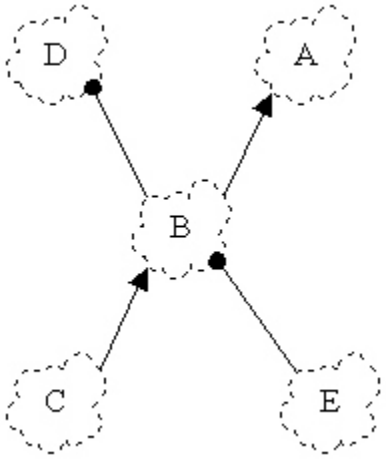
> Containment is the "has-a" relationship; it should be used when instances of a class in some sense contain instances of another class. The containment might be physical or logical. Inheritance is the "is-a" relationship; it should be used when an instance of a class can be thought of as being an instance of another class.

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

> The ability to write code in terms of something general, such as a geometric shape, and then use that code with instances of derived classes that are specialized, such as circles and triangles.

Problem 2 (5 points):

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




```
class A {};

class E {};

class B: public A {
     E itsE;
     };

class C: public B {};

class D {
     B itsB;
     };
```

Problem 3 (20 points):

```
#include <iostream.h>
#include "string.h"

class ReplStr {
  public:
     ReplStr() {}
     ReplStr(String s, int n) {
          itsStr = s.Repl(n);
          }
     int Length() { return itsStr.Length(); }

     String GetString() { return itsStr; }

     int operator==(ReplStr rhs) {
          return itsStr == rhs.itsStr;
          }

     int operator!=(ReplStr rhs) {
          return !(this->GetString() == rhs.itsStr);
          }

     ReplStr operator+(ReplStr rhs)
     {
          String r = GetString() + rhs.GetString();

          return ReplStr(r, 1);
     }
```

```
  private:
     String itsStr;
     };

ostream& operator<<(ostream& o, ReplStr s)
{
     o << s.GetString();
     return o;
}
```

(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.

> The choice to represent a `ReplStr` as a `String` makes for a simple implementation but could be very inefficient. If nothing else, construction of the `String` could be deferred until the first time `GetString` is called. (Note that length can be calculated given only the base and replication count.)

Problem 4 (5 points):

Fully implement classes `Bicycle`, `Tricycle`, and the function `CountWheels`.

```
class Cycle {
  public:
    Cycle(char *owner) {
        itsOwner = new char[strlen(owner)+1];
        strcpy(itsOwner, owner);
                }

        virtual int GetNumWheels() = 0;
        char *GetOwner() { return itsOwner; }
  private:
        char *itsOwner;
        };

class Bicycle: public Cycle {
  public:
        Bicycle(char *owner) : Cycle(owner) {}
        virtual int GetNumWheels() { return 2; }
        };

class Tricycle: public Cycle {
  public:
        Tricycle(char *owner) : Cycle(owner) {}
        virtual int GetNumWheels() { return 3; }
        };

int CountWheels(Cycle *cycles[])
```

```
{
        int wheels = 0;

        for (Cycle **cp = cycles; *cp; cp++)
                wheels += (*cp)->GetNumWheels();

        return wheels;
}
```

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.

```
flip([],[]).

flip([X1,X2|XS],[X2,X1|NewXS]) :- flip(XS, NewXS).
```

Problem 6 (5 points):

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

```
append([],L,L).
```

The list `L` appended to the empty list is the list `L`.

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

The list `L2` appended to the list having head `X` and tail `L1` is the list with head `X` and tail `L3` where `L3` is the result of appending `L2` to `L1`.

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.

```
trim(L, SL, NL) :- append(SL, NL, L).
trim(L, SL, NL) :- append(NL, SL, L).
```

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.

```
maxint([X],X).

maxint([X1,X2|XS], M) :-
    X1 > X2, maxint([X1|XS], M).

maxint([X1,X2|XS], M) :- X1 =< X2, maxint([X2|XS], M).
```

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

```
sum_check(0,[]).
sum_check(Sum, [H|T]) :-
     sum_check(TSum, T), Sum is H + TSum.
```

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.

```
three_eq(X,X,X).
```

Problem 10 (5 points):

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

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

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

```
knows(mary,X), shift(mary,S), \+shift(X, S).
```

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

```
knows(X,Y) :- shift(X,S), shift(Y,S), X \== Y.
```

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

```
by_shift :- report, fail.

report :- write('Day:'), nl, shift(day), nl.
report :- write('Night: '), nl, shift(night), nl.

shift(X) :- shift(P,X), write('   '), write(P).
```

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.

```
fun allsame([]) = false
 |  allsame([x]) = true
 |  allsame(x1::x2::xs) = x1 = x2 andalso allsame(x2::xs)
```

Problem 12 (2 points):

Consider this function definition:

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

What types would ML infer for:

```
f?   int * (int -> int list) -> int list

g?   int -> int list

x?   int
```

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.

```
procedure main()
    lines := []

    while push(lines, read())

    while write(get(lines))
end
```

Problem 14 (20 points):

If you don't know this by now, I doubt that telling you one more time will help...

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 surprising result. What was that result?

It was detected that a program would not terminate.

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

ML, Icon, C, C++, and Prolog

(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.

```
int strlen(char *p)
{
     if (!*p)
          return 0;
     else
          return 1 + strlen(p + 1);
}
```

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

Schonfinkel. (Curry came along later.)

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

Windows NT

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

Did you?

CSc 372, Fall 1996
Mid-Term Examination Solutions

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")
```

```
int * int * (bool * int) * string
```

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

```
int list list
```

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: `string -> int`
b: `string`
c: `(string -> int) -> int list`
The type of the result produced by `f`: `bool`

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`.

```
fun f a b c d = a + b + c + d = 1 (* A. Freshwater *)
```

- or -

```
fun f 1 2 3 4 = true
```

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.

Most solutions took this form:

```
fun repl(s, 0) = ""
 |  repl(s, n) = s ^ repl(s, n-1);

fun ints_to_strs([], _) = []
 |  ints_to_strs(x::xs, s) = repl(s, x) :: ints_to_strs(xs, s)
```

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)

```
fun len(L) = size(reduce(op^,reduce(op@, L))) (* P. Holland *)
```

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`.

```
exception OddLength;

fun pair([]) = []
 |  pair([x]) = raise OddLength
 |  pair(x1::x2::xs) = (x1,x2)::pair(xs)
```

Problems 7 and 8: (26 and 10 points, respectively)

```
class Food {
  public:
    virtual int GetUnits() = 0;
    };

class Burger: public Food {
  public:
    int GetUnits() { return 25; }
    };

class Fries: public Food {
  public:
    int GetUnits() { return 15; }
    };
```

```
class Eater {
  public:
    Eater(int capacity)
     : itsMax(capacity), itsBurps(0), itsValue(0) {}

    void Eat(int units) {
        itsValue += units;
        while (itsValue >= itsMax) {
            cout << "burp!" << endl;
            itsBurps++;
            itsValue -= itsMax;
            }
        }

    void Eat(Food *fp) {
        Eat(fp->GetUnits());
        }

    int BurpCount() const { return itsBurps; }

  private:
    int itsMax, itsBurps, itsValue;
    };

ostream& operator<<(ostream& o, Eater& E)
{
    o << "Eater at " << &E << " has burped " << E.BurpCount()
           << " times" << endl;

    return o;
}
```

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?

An object is an instance of a class.

What is the purpose of a constructor?

To initialize an object.

What is the purpose of a destructor?

To reclaim resources acquired during the lifetime of the object.

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

"Zero" is a reasonable answer, but with proper rationale the answer "any number" is also acceptable.

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.

> Using an inline member function maintains encapsulation but without any performance penalty.

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

> No, the purpose of member functions is to provide the required behavior for the class. Data members merely support member functions.

Describe a benefit provided by using inheritance.

> Being able to treat an object as being something more general than it really is.

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)

> With pattern matching a non-exhaustive match can indicate a situation where a usage of `hd` could produce an exception. Pattern matching can also be argued for on the basis of efficiency and clarity.

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

> The intended response was "`class X {};`" but a number of persons observed that "`typedef char X;`" or "`#define X char`" was enough.

Could C++ be used effectively for functional programming? (5 points)

> This could be argued successfully either way.

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)

```
int strcmp(char *a, char *b)
{
    if (*a == '\0' && *b == '\0') return 0;
    else if (*a == '\0' || *b == '\0') return 1;
    else if (*a == *b) return strcmp(a+1, b+1);
    else return 1;
}
```

CSc 372, Fall 1996
Final Examination Solutions

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.

ML: Type deduction, partial application of functions, no variables

C++: Supports an object oriented paradigm, type extensibility, templated functions.

Icon: Expression failure, string scanning, a compact but expressive set of built-in functions.

Prolog: Procedures (predicates) with multi-faceted behavior, the notion of unification, expressions represented as trees that are evaluated on demand.

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.

ML: A partial application can be used to create a more specialized function from a more general function.

C++: Encapsulation of data in objects allows the data to change without requiring modification to code that uses those objects.

Icon: The set of built-in functions is compact enough that I'm personally able to write Icon programs of a significant size without needing to consult any reference material.

Prolog: The multi-faceted behavior of predicates allows one predicate to serve several purposes. For example, `member/2` can be used to test for membership, generate elements, and more.

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.

```
permute([A],[A]).
permute([A,B],[A,B]).
permute([A,B],[B,A]).
permute([A,B,C],[A,B,C]).
permute([A,B,C],[A,C,B]).
permute([A,B,C],[B,A,C]).
permute([A,B,C],[B,C,A]).
permute([A,B,C],[C,A,B]).
permute([A,B,C],[C,B,A]).
```

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?

> The ML function provides only half the functionality of the predicate, which can also map from given decimal numbers to their Roman equivalent.

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.

```
sum_ints([], 0).
sum_ints([H|T], Sum) :- integer(H), sum_ints(T, TSum),
                            Sum is H+TSum, !.
sum_ints([_|T], Sum) :- sum_ints(T, TSum), Sum is TSum.
```

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`.

```
assemble(L, Segments) :- member(A,Segments), member(B,Segments),
    append(A,B,L).
```

Problem 7 (10 points):

```
cfa([], _, []).
cfa([Amt|Amts], Coins, [Change|MoreChange]) :-
    make_change(Coins, Amt, Change, Left),
    cfa(Amts, Left, MoreChange).
```

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.

```
procedure extract(s, i, j)
    return split(split(s, '/')[i], ':')[j]
end
```

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.

```
procedure main()
    while line := read() do
        line ? {
            tab(upto('a')) & tab(upto('b')) & tab(upto('c')) &
            write(line)
            }
end
```

Problem 10 (6 points)

Write an Icon procedure `revby2(s)` that reverses the string on a character pair-wise basis and returns the resulting string. **NOTE: Your solution must use string scanning. In particular you may not use string subscripting, sectioning, or the * operator.**

```
procedure revby2(s)
    r := ""

    s ? {
        tab(0)
        while r ||:= move(-2)
        pos(1) | fail
        }

    return r
end
```

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`.

```
procedure main()
    every f("am")
    every f("pm")
end

procedure f(m)
    every write(right(12 | (1 to 11), 2, " "), ":",
       right(0 to 59, 2, "0"), m)
end
```

More concisely:

```
procedure main()
    every m := ("am"|"pm") &
        write(right(12 | (1 to 11), 2, " "), ":",
            right(0 to 59, 2, "0"), m)
end
```

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.

```
procedure main()
    L := []
    while push(L,read())
    while write(pop(L))
end
```

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.

```
fun sum3(a,b,c) = a+b+c:int

fun samesums([]) = true
 |  samesums([t]) = true
 |  samesums(t1::t2::ts) =
        sum3(t1) = sum3(t2) andalso samesums(t2::ts);
```

Problem 14 (9 points)

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

```
#include <iostream.h>

class X {
  public:
    X(int a, int b) : itsSum(a+b) {}
    int f() { return itsSum; }
    void make5();
  private:
    X() :itsSum(0) {}
    int itsSum;
    };

void X::make5()
{
    X a, xs[3];
    X *p = new X();

    cout << a.f() + p->f() + xs[0].f() + xs[1].f() + xs[2].f()
       << endl;
}

ostream& operator<<(ostream& o, X& x)
{
    o << "an X at " << &x;
    return o;
}
```

EC 1 (5 points):

Name five programming languages that originated before 1985.

ML, C++, Icon, Prolog, and C.

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.

ML—Robin Milner et al.
C++—Barne Stroustrup
Icon—Ralph Griswold
Prolog—Alain Colmeraurer
Java—James Gosling

EC 3 (1 point):

What is the instructor's favorite programming language?

I guess that if I've got a favorite it would have to be Icon, but it really depends on the

situation. Any answer here was worth a point.

EC 4 (1 point):

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

Windows NT

EC 5 (2 points)

Languages can be grouped according to various aspects of the language. Group the languages we studied according to their type checking philosophy.

One possible grouping is that type-checking is done at compile time in ML and C++, but at run time in Icon and Prolog.

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.

```
procedure main()
    while line := map(read()) do
        if *(line -- 'abcdef') = 0 then
            write(line)
end
```

EC 7 (1 point)

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

Any answer here was worth a point.

EC 8 (2 points)

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

For me, it's probably ML's ability to support partial applications of functions.

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.

Automatic memory management, function bodies must appear in the class definition, and there can be no global functions or data.

EC 10 (2 points)

Why are static class members an essential element of Java?

Because there are no global functions or global data.

EC 11 (1 point)

What is the Prolog 1000?

A compilation of applications implemented in Prolog.

CSc 372, Spring 1997
Mid-term Examination Solutions

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])
```

```
int list * real list
```

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

```
int list
```

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

Invalid: `[3, "z"]` attempts to form a heterogeneous list

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

```
('a list -> int) * (string -> int) list
```

Problem 2: (2 points)

Consider this ML function definition: `fun f(x) = [x 1, 2]` What type will be deduced for `f`?

```
(int -> int) -> int list
```

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: int`
`b: int list`
`c: int`
The return type of `f: int`
The return type of `g: int list`

Problem 4: (3 points)

Write a function `f` that has the type `int -> int list -> bool`.

```
fun f 1 [1] = true
```

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`.

```
fun tossbig([], _) = []
 |  tossbig(s::ss, n) =
        if size(s) < n then s::tossbig(ss, n)
                       else tossbig(ss,n)
```

Or, a non-recursive solution using `filter` from the slides:

```
fun tossbig([], _) = []
 |  tossbig(L, n) = filter(fn(s) => size(s) < n, L)
```

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.

```
fun samesums [] = true
 |  samesums [_]  = true
 |  samesums ((a,b)::(c,d:int)::ts) =
     a+b = c+d andalso samesums((c,d)::ts)
```

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.**

When this problem was written, an answer like this was envisioned:

```
fun block(w,h) =
    let
        fun row _ =
           implode (map (fn(x) => "x") (m_to_n(1,w))) ^ "\n"
    in
        print(implode ((map row) (m_to_n(1,h))))
    end
```

However, some persons observed that the solution could be simplified with `repl`, from the second assignment:

```
fun block(w, h) = print(repl(repl("x", w) ^ "\n", h))
```

If I had considered that `repl` might be used, I would have specifically excluded it, but I

since I did not, I considered solutions using `repl` to be perfectly acceptable, even though that essentially side-stepped the purpose of the problem.

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.

```
fun c_block w h = block(w,h)
```

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. ...

```
class TV {
    ...
  private:
    int itsFavs[84];
    int itsCurChan;
    int itsCurVol;
    };

TV::TV()
  : itsCurChan(2), itsCurVol(5)
{
    for (int i = 2; i < 84; i++)
        itsFavs[i] = -1;
}

int ckRange(int low, int high, int val)
{
    if (val < low || val > high)
        return 0;
    else
        return 1;
}

void TV::SetVol(int vol)
{
    if (!ckRange(0, 10, vol))
        return;

    itsCurVol = vol;
}

void TV::SetChan(int chan)
{
    if (!ckRange(2, 83, chan))
        return;

    itsCurChan = chan;
}

void TV::SetFav()
{
```

```
    itsFavs[itsCurChan] = itsCurVol;
}

void TV::SetFav(int vol)
{
    if (!ckRange(0, 10, vol))
        return;

    itsFavs[itsCurChan] = vol;
}

void TV::NextFav()
{
    int i;
    for (i = itsCurChan+1; i <= 83; i++)
        if (itsFavs[i] != -1) {
            SetVol(itsFavs[i]);
            SetChan(i);
            return;
            }

    for (i = 2; i < itsCurChan; i++)
        if (itsFavs[i] != -1) {
            SetVol(itsFavs[i]);
            SetChan(i);
            return;
            }
}

void TV::Status()
{
    cout << "Channel is " << itsCurChan << ", volume is "
        << itsCurVol << endl;
}
```

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`.

```
int operator%(const String& s, char c)
{
    char *p = strchr(s.GetPtr(), c);

    if (p == 0)
        return -1;
    else
        return p - s.GetPtr();
}
```

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?

An object is an instance of a class.

(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.

The responsibility of a constructor is to initialize an object. The destructor's responsibility is to surrender resources acquired during the lifetime of the object.

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

In-line functions provide access that is just as fast as directly referencing data members, but without loss of encapsulation.

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.

```
class Ring: public Shape {
    public:
        Ring(double inner, double outer)
        : itsInner(inner), itsOuter(outer) {}

        double Area() {
            return itsOuter.Area() - itsInner.Area();
            }

        double Perimeter() {
            return itsOuter.Perimeter();
            }

        void Print(ostream& o) {
            o << "Ring; area = " << Area() << ", perim = "
                << Perimeter() << endl;
            }

    private:
        Circle itsInner, itsOuter;
        };
```

No changes are required in `SumOfAreas` or `Biggest`.

CSc 372, Spring 1997
Final Examination Solutions

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?

That expression has no meaning in Prolog based on the material covered but in Icon it's an expression that has the result sequence `{[10,20], [10,30]}`.

What is the type of the following ML expression?

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

```
(('a list -> int) list * int list) list
```

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
```

```
fun map F [] = []
 |  map F (x::xs) = F(x)::(map F xs)
```

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

A programming method where one creates a system of entities and orchestrates interaction between them.

Cite a fundamental benefit of inheritance in C++.

Objects can be treated as being something more general than they really are.

What's unusual about this Prolog predicate?

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

The cut comes after `fail` and is thus never reached.

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

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

A list that can be divided into two identical lists. Here's one: `[1,2,3,1,2,3]`.

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]).
```

```
ABABACC
yes
```

Write the `append/3` predicate.

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

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

Prolog and Icon have several things in common. Among them: automatic memory management, backtracking, a heterogeneous list type, and a FORTRAN-based first implementation.

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.

I was bit worried about putting a question of this size on the test and I tried to compensate for that with a very direct advance hint via mail and by citing a possible decomposition in the exam itself. I liked the fact that the problem required use of many elements of Icon and was practical rather than contrived. But, you know what's said about the best laid plans and the fact is that most persons didn't do very well on this problem.

To help you understanding the grading, I thought of it as thirty point problem with five parts: (1) parsing of the "case..." line, (2) parsing of the configuration, (3) verification of

the configuration's length and matings, (4) verification that the cables used were drawn from the ones supplied, (5) a main program to tie things together. Each part was worth six points and there was a one point bonus for writing down anything. When grading I deducted only for major flaws and omitted elements—code that appeared to be in the ballpark got full credit. Most persons ended up in the mid to low 20s. There was a 28, two 29s, and one 31.

```
link split

procedure do_case(s)
    cabs := []
    s ? {
        tab(upto('\['))
        pieces := split(tab(0), '[],\'')
        e2 := pull(pieces)
        len := pull(pieces)
        e1 := pull(pieces)

        every i := 1 to *pieces by 3 do
            put(cabs, pieces[i:i+3])
        }

    return [cabs,len,e1||e2]
end

procedure do_config(s)
    cabs := []

    s ? while e1 := move(1) do {
            len := *tab(many('-'))
            e2 := move(1)
            put(cabs, [e1,len,e2])
            }


    return cabs
end

procedure config_ok(cabs, len, ends)
    if (cabs[1][1] == ends[1]) | (cabs[-1][3] == ends[2]) then
        fail

    every len -:= (!cabs)[2]

    if len <= 0 then
        return
end

procedure sets_ok(In,Out)
```

```
    t := table(0)
    every t[c_to_rep(!In)] +:= 1
    every t[c_to_rep(!Out)] -:= 1

    if !t < 0 then
        fail
    else
        return
end


procedure c_to_rep(cable)
    cable := sort(cable)
    return map(cable[1] || cable[2] || cable[3])
end

procedure main()

    while data := do_case(write(read())) do {
        confln := write(read())
        if confln[1] ~== "C" then {
            config := do_config(confln)

            if not config_ok(config, data[2], data[3]) |
               not sets_ok(data[1], config) |
               find("MM"|"FF", confln) then
                    write("**ERROR**")
            }
        write(read())
        }
end
```

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([],[]).
sort(L,[X|Sorted]) :-
    min(X,L),
    getone(X, L, Left),
    sort(Left,Sorted).
```

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.

```
notallsame([X,X,X]) :- !, fail.
notallsame(_).

hexdig(X) :- member(X,[0,1,2,3,4,5,6,7,8,9,a,b,c,d,e,f]).

hexprint :- hexdig(D1), hexdig(D2), hexdig(D3),
          notallsame([D1,D2,D3]),
          write(D1), write(D2), write(D3), nl, fail.
hexprint.
```

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.

```
palsum([]).
palsum([_]).
palsum([H|T]) :- append(Middle,[Last],T), sum(H,Sum),
sum(Last,Sum),
                 palsum(Middle).
```

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.

A predicate can perform different functions depending on what's instantiated. The `member` predicate is a simple example:

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

Member can (1) test to see if a value is in a list (2) generate all values in a list (3) generate lists containing a given value. The first two in Icon:

```
procedure is_member(x, L)
    if x === !L then return
                else fail
end

procedure members(L)
    suspend !L
end
```

Problem 7 (4 points):

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

Answer-wise anything that was coherent earned all four points on this problem.

Personally, I find them all them pretty interesting, but if I had to pick one that's the most interesting to me at the moment it would have to be Prolog.

The responses:

Icon	*12.5*
Prolog	*11.5*
C++	*4*
ML	*0*

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.

As in the last problem, any sort of coherent response got all four points.

Of the group I found Prolog to be the most difficult. For one thing, I have a hard time visualizing a Prolog computation that uses a lot of backtracking.

The responses:

Prolog	*21*
C++	*3*
ML	*3*
Icon	*1*

Historical note: Last semester a show of hands poll at the final showed about a 50/50 split between Icon and Prolog with one for ML and zero for C++ as the most difficult language.

Problem 9 (3 points):

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

```
fun f x _ = x
```

Problem 10 (4 points):

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

At the time of the test I hadn't written a solution for `dups` and while driving to campus I convinced myself that the problem was harder than I first thought it was. I therefore added `biggest(L)` as an alternative on the spot, but `dups` was actually very simple:

```
fun dups(s) =
    let
        fun dups1([]) = []
         |  dups1([c]) = [c]
         |  dups1(c1::c2::cs) =
            if c1 = c2 then
                dups1(c2::cs)
            else
                c1::dups1(c2::cs)
    in
        implode(dups1(explode(s)))
    end
```

Here's `biggest`:

```
exception Empty;
fun biggest [] = raise Empty
 |  biggest [x] = x:int
 |  biggest (x1::x2::xs) =
        if x1 > x2 then
            biggest(x1::xs)
        else
            biggest(x2::xs)
```

`biggest` can also be done with `reduce`, but I'll leave you to think about that one.

Problem 11 (8 points):

Implement two C++ classes: `MList` and `PrintableMList`. ...

```
#include <iostream.h>

class MList {
  public:
    MList(int maxVal)
     : _maxVal(maxVal), _count(0), _closed(0) {}
    void operator<(int value) {
        if (!_closed && value <= _maxVal)
            _values[_count++] = value;
        }

    void close() { _closed = 1; }
```

```
    int size() { return _count; }

  protected:
    int _maxVal, _count, _closed, _values[100];
    };

class PrintableMList : public MList {
  public:
    PrintableMList(int maxVal)
    : MList(maxVal) {}
    void print() {
        for (int i = 0; i < _count; i++)
            cout << _values[i] << " ";
        cout << endl;
        }
    };
```

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.

Limbo and Java

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?

According to an interview with A. Joseph Hoane, Jr. that can be found at `www.chess.ibm.com/meet/html/d.4.5.a.html`, it's written in C.

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.

```
% cd /home/cs372/Handouts
% grep ^%%Page: *.ps | wc -l
     513
%
```

EC 4 (1 point):

Name a language that is an ancestor of Icon.

SNOBOL4

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.

The copy parked in `/home/cs372/Prolog/Prolog-1000` appears to have 502 entries, but is said to be "preliminary". However, I think that in class I might have said that the list had more than one thousand entries and therefor, either (a) or (c) earned a point.

EC 6 (1 point):

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

A bowling pin. (I put this question here as a token reward for those students who kept me from being lonely during office hours!) Other responses were baseballs, footballs, and a tennis racket.

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.

My answer:

```
*[[]]||*[]
```

Mr. Klein was the only person who answered this question correctly. His answer:

```
*"\\"||*""
```

EC 8 (1 point):

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

If I only remember one thing from teaching CSc 372 this time around it will be nearly losing my mind trying to think up some decent problems for the first C++ assignment.

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"`.

This is a classic example of omitting a key element of a question. I should have said "...prints, one per line, the letters..." The answer I had in mind was this,

```
write(!&lcase) & 1 = 0
```

but a number of persons observed that the specifications could be met with this,

```
write(&lcase)
```

and that earned full credit.

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.

```
procedure allsame(s)
    *cset(s) = 1 & return
end
```

EC 11 (1 point): (Written on the board...)

Sometimes in Tucson the only difference between the time and the temperature is a _____.

colon (For example, it might be 102 at 1:02.)

CSc 372, Fall 2001
ML Examination Solutions

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)
```

*int * int list * real*

```
(reduce op+)
```

int list -> int

```
explode o rev o implode
```

Not valid – `implode` produces a string but `rev` requires an `'a list`

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

(string -> int) list list list

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

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

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

*int * int -> int*

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

int -> int

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

*(''a * ''b) * (''a * ''b) -> bool*

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

*int * int -> (string -> int) * ('a list -> int)*

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

(int -> int) -> int -> int

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(a, _, c) = [a-c, a+c];
```

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

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

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

```
fun f2(10) = true
 |  f2(5)  = true
 |  f2(_)  = false
```

Problem 4: (5 points)

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

```
fun map F [] = []
 |  map F (x::xs) = F(x)::(map F xs)
```

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.

```
fun abslist(L) = map (fn(x) => if x < 0.0 then ~x else x) L
```

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
```

Change this:

```
fun gather(L, limit) =
```

to this:

```
fun gather([], _) = []
 |  gather(L, limit) =
```

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.

```
fun doubler L =
  let
    fun f([]) = []
     |  f(c::cs) = c::c::f(cs)
  in
    (map (map (implode o f o explode))) L
  end

val quadrupler = doubler o doubler
```

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.

```
val genlist = map ((map (fn(_) => 1)) o iota)
```

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).

```
val genlist_inv = map sum
```

An acceptable answer is to use `length` instead of `sum`,

```
val genlist_inv = map length
```

but if you try it out with the interpreter, you'll find that you get an error about type variables not being generalized.

Problem 8c: (2 points)

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

```
int list list -> int list
```

Problem 9: (18 points)

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.

```
fun tacdel(fname) =
     let
          val bytes = read_all_bytes(fname)
          fun lmapper(s) =
               if member(#"@", explode s) then "<D>" else s
          val lines = map lmapper (rev (split #"\n" bytes))
     in
          print(concat(ien(lines, 1, "\n")))
     end
```

CSc 372, Fall 2001
Icon Examination Solutions

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.

```
procedure main()
    L := []
    while line := read() do
        if line ? find("@") then
            push(L, "<@>")
        else
            push(L, line)
    every write(!L)
end
```

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.

```
procedure main(a)
    x := read_file(a[1])
    y := read_file(a[2])

    if *x ~= *y then stop("Diffs")
    every i := 1 to *x do
        if x[i] ~== y[i] then
            stop("Diffs")
end
```

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`.

A straightforward solution is this:

```
procedure main()
    t := table(31)
    every t[4|6|9|11] := 30
    t[2] := 28
    every m := 1 to 12 do
        every d := 1 to t[m] do
            every y := 1 to 99 do {
                s := m || "/" || d || "/" || y
                write(reverse(s) == s)
                }
end
```

With a little thought about the possibilities, the program can simplified:

```
procedure main()
    every m := 1 to 12 do
        every d := (1 to 9) | 11 | 22 do
            every y := 1 to 21 do {
                s := m || "/" || d || "/" || y
                write(reverse(s) == s)
                }
end
```

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.

```
procedure main()
    p := table()
    while line := read() do {
        if line == "." then
            break
        reverse(line) ? {
            price := reverse(tab(upto(' ')))
            if price[-1] == "c" then
                price := price[1:-1] * .01
            else
                price := price[2:0] * 1.0
            tab(many(' '))
            item := reverse(tab(0))
            p[item] := price
            }
        }

    total := 0
    while total +:= p[read()]
    write("$", total)
end
```

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.

```
procedure intmem(i, L)
    return integer(!L) = i & &null
end
```

Problem 6: (10 points)

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

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 ==.

```
procedure main()
    sum := 0
    while line := read() do
        line ? while tab(upto(&digits)) do
            sum +:= tab(many(&digits))

    write(sum)
end
```

Problem 7: (6 points)

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

Expressions can produce zero, one, or many results.

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.

Some examples:
`t := table([])` *doesn't mix well with* `push` *and* `put`
Semicolon insertion can lead to surprises
No well-defined and documented library to do things like reversing a list

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?

The limited memory space (< 64k of compiled code) for the initial UNIX implementation and relative difficulty of writing those functions in C.

Problem 8: (8 points)

Fill in the blanks:

The instructor described the function _`move`_ as being a "lone wolf". He said that _`tab`_ "works well with others".

The result of a successful comparison in Icon is the *right hand operand*.

We studied a total of _*8*_ string scanning functions. Of those, two changed _*the position*_ and _*five*_ of them returned a _*position*_. _`pos`_ is one-of-a-kind.

EXTRA CREDIT SECTION (one point each)

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

```
][ .every (?"xxx" || !"xxx" || *"xxx");
   "xx3"  (string)
   "xx3"  (string)
   "xx3"  (string)
```

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

```
][ s := "whm" & string(cset(s[1:4]))[1:-2];
   r := "h"  (string)

][ s := "yourname" & string(cset(s[1:4]))[1:-2];
   r := "o"  (string)
```

(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)

The unary \ and / operators

It can be argued that the the augmented operators, such as `+:=`, are syntactic sugar.

Two elements we didn't talk about, the `repeat` and `case` expressions, are reasonably thought of as syntactic sugar.

(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`"?

It produces no output.

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

```
][ "" ? tab(2);
Failure

][ "" ? move(2);
Failure
```

CSc 372, Fall 2001
Prolog Examination Solutions

Problem 1: (5 points)

Show an example of each of the following:

A fact: `p.`

A rule: `p :- q.`

A query: `p.`

A clause: `p.`

An atom: `p`

Problem 2: (2 points)

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

Facts and rules are 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).
```

False—it only succeeds if a value is the last element of the list.

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.
```

False—`Sum is Sum + 1` fails.

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([ ], _, 0).
sumval([X|T], X, Sum) :- sumval(T, X, TSum), Sum is X + TSum, !.
sumval([_|T], X, Sum) :- sumval(T, X, Sum).
```

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.

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.

```
sumvals([], _, 0).
sumvals([H|T], L, Sum) :- member(H, L), sumvals(T, L, TSum),
                          Sum is H + TSum, !.
sumvals([_|T], L, Sum) :- sumvals(T, L, Sum).
```

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(L,L).
```

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(Val, N, List) :- repl(Val, N, Vals),
                        append(Vals, _, List), !.

consec(Val, N, [_|T]) :- consec(Val, N, T).
```

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(L,[A,B,C]) :-
     getone(A,L,R), getone(B,R,[C]), A =< B, B=< C, !.
```

Problem 10: (20 points)

In this problem you are to write a predicate `inventory/0` that does an inventory calculation for a fruit stand. [...]

```
inventory :- fruit(F), get_qty(F,Q), cost(F,C), TC is Q*C/100,
             format('~p: ~p at ~p = $~p~n', [F,Q,C,TC]), fail.
inventory.

get_qty(F,Q) :- qty(F,Q), !.
get_qty(_,0).
```

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 = 4
     B = 1
     C = [2,3]

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

This query fails because `B` can't be unified with both `1` and `2`.

EXTRA CREDIT SECTION (one point each)

(a) What is the Prolog 1000?

A list of significant applications written in Prolog and related languages.

(b) In what country was Prolog developed?

France

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

Japan, with its Fifth Generation project.

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

FORTRAN

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

Silence

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

The correct specification is `append(?L1, ?L2, ?L3)`. All, some, or none of the arguments might be specified.

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

The variable is question is used only once. That might indicate a misspelled name or other error.

(h) In a Prolog library you see these two predicates: `get_chr(+Number, -Char)` and `get_ord(+Char, -Number)`. What's odd about that?

With Prolog, one predicate can perform both calculations.

CSc 372, Fall 2001
Final Examination Solutions

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 "`#`".

```
(defun change ()
    (interactive)
    (goto-char 1)
    (let (ch)
        (while (not (eobp))
            (setq ch (char-after (point)))
            (delete-char 1)
            (cond
                ((letterp ch) (insert ?x))
                ((digitp ch) (insert ?#))
                (t (insert ch)))
            )
        )
)
```

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

Write a function `ruler`, bound to `ESC-R`, that inserts a "ruler" before the current line.

```
(defun ruler ()
    (interactive)
    (let ((w (1- (window-width))) (current (point)) start end)
        (forward-line 0)
        (setq start (point))
        (let ((n (/ w 10)) (i 1))
            (while (<= i n)
                (insert "         " (int-to-string i))
                (setq i (1+ i)))
            (insert "\n")
            (setq i 1)
            (while (<= i w)
                (insert (int-to-string (% i 10)))
                (setq i (1+ i)))
            (insert "\n")
            )
        (read-char)
        (delete-region start (point))
        (goto-char current)
        )
    )

(global-set-key "\eR" 'ruler)
```

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.

```
(defun longest()
    (interactive)
    (goto-char 1)
    (let ((max -1) (last (point)) (maxpos 1) len)
        (while (not (eobp))
            (forward-line 1)
            (setq len (- (point) last))
            (cond
                ((> len max) (setq max len)
                             (setq maxpos last)))
            (setq last (point))
            )
        (goto-char maxpos)
        )
    )
```

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.

```
(defun marquee (text)
    (interactive "sText? ")
    (get-empty-buffer "*m*")
    (switch-to-buffer "*m*")
    (insert-char ?\n (/ (window-height) 2))
    (setq pos (point))
    (insert text)
    (let (ch)
        (while t
            (goto-char pos)
            (setq ch (char-after pos))
            (delete-char 1)
            (end-of-line)
            (insert ch)
            (sit-for 0 70)
            )
        )
    )
```

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.

One thing that comes to mind for me is to add object-orientation so that instead of a big pile of functions we could have classes like `Buffer`, `Window`, etc. Buffer-specific operations might be methods of the `Buffer` class, for example. It is the case however, that a simpler module-based approach, somewhat like ML's structures, would provide sufficient grouping to make the task of finding functions more manageable.

For a second item, I think that Icon's idea of keywords—things like `&subject` and `&pos` might be useful. For example, imagine `&point`. To move forward one might say `&point +:= 1`. `&point := 1` would go to the beginning of the buffer and `&point := 0` would go to the end of the buffer. Instead of several functions to adjust the point, one would simply change the value of `&point`.

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.

```
fun rm_prefix([],L) = L
 |  rm_prefix(_::ps, _::xs) = rm_prefix(ps, xs);
```

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.

```
fun listeq(L1, L2) = L1 = L2;
```

Problem 8: (8 points)

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

```
procedure extsort(L)
    return swap(sort(swap(L)))
end

procedure swap(L)
    L2 := []
     every w := split(!L, ".") do
          put(L2, w[2] || "." || w[1])
    return L2
```

```
end
```

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`.

```
find(N,[A,B,C]) :- iota(N,L), getone(A,L,R), getone(B,R,R2),
    getone(C, R2, _), N is A * B - C.
```

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.

When describing the idea of this course to someone I often use `member` in Prolog as an example of an interesting way to do something in another language:

In Prolog I might define a *predicate* called `member` that tests for membership in a list. Here is the definition of member:

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

That says that `X` is a member of any list of which it is the first element or any list that it is a member of the tail. An interesting thing about Prolog is that because predicates describe relationships, we can use that same predicate to generate the elements in a list:

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

Problem 11: (10 points)

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?

Before Java's quick rise to popularity I was more skeptical about the prospect of a single language being dominant but Java, with all its problems, has nonetheless transformed much of the programming landscape. I have to think that a language of comparable complexity that addresses Java's weaknesses might have a chance of becoming dominant across many, if not most application areas.

It seems that a language with a potential of dominance would have to come from academia or the open source community. Vendors have a bad track record of supporting ideas from other vendors, no matter how good they are. Ideas that emerge from neutral parties often have a greater likelihood of widespread adoption.

In terms of language characteristics, it seems that object-orientation, a conventional syntactic structure, automatic memory management, and the potential of efficient execution are all musts.

With regard to opposing forces, it seems to be the case that successful software systems grow and grow. A dominant language would probably grow and grow until at some point it became clear that a language with a subset of the capabilities would serve just as well in many cases and perhaps excel in other areas. A crisp new language would perhaps attract those who like to be different and perhaps the cycle would repeat itself.

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.

A big advantage is that you don't have work around reserved words when choosing variable names. I've many times wanted to use the variable "`to`" in an Icon program but that conflicts with `to-by`.

In terms of negatives, there is the potential for writing code like the above, but that's rarely a problem in practice. Also, it's difficult to write ad-hoc source code analysis tools to deal with programs written in such languages.

(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?

There can be a tendency to have a polymorphic operator cover too much ground. For example, should Icon's `!` operator handle numbers as well? Should the `delete` function operate on lists as well? There can also be a problem when it comes to naming an operation but using an operator instead of a function is one way to avoid the question of what-to-call-it.

(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.

Python is a language that has heterogenous lists as well as tuples. Python's tuples are immutable—they can't be changed—and in *Learning Python* that immutability is cited as the best reason for the co-existence of the two mechanisms. I have used Icon's lists as tuples in various situations and there is something that feels a little bit uncomfortable about it, but I can't put my finger on it—maybe it is mutability.

(D) What do Java's exception handling mechanism and Icon's failure mechanism have in common?

The common element is the situation of an expression for which there's no reasonable result. In Icon an out of bounds array index fails; in Java an exception is thrown. Exceptions are far more general but much more cumbersome to use. It would be nice to have a mechanism with the generality of exceptions and Icon's conciseness of expression.

(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.)

The common elements that I see are (1) simple I/O, (2) good string handling, (3) data types like lists and tables, and (4) a library that's sufficiently well-designed that you don't need to look up the arguments for every other function.

(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?

One way to think about built-in functions is which can be implemented in the language and which cannot. For example, there's no way to exactly implement Icon's `push/pop/pull/...` functions with the other mechanisms of the language—you can't write those functions in Icon.

Another consideration is efficiency. It may possible to implement a function in Lisp, but if the function is long-running and heavily used, it might be better written in C.

A third issue is complexity. A function may be sufficiently complicated that it is impractical to write in the implementation language.

EXTRA CREDIT SECTION (one point each)

(a) What's the difference between a hack and programming technique?

When you do it, it's a programming technique. When somebody else does it, it's a hack.

(b) Order these languages from oldest to youngest: Icon, Java, Lisp, ML, Prolog.

Lisp, Prolog, ML, Icon, Java

(c) Write an expression that is valid in three of the languages we studied but with a notably

different interpretation in each.

Good question!

(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%.)

280 messages

(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?

Python

(f) What is the type of `rm_prefix` (problem 6)?

```
'a list * 'b list -> 'b list
```

CSc 372, Fall 2006
Mid-Term Examination Solutions

TOP SECRET

Circumstances have still not allowed two students to take a make-up for this exam. Guard these solutions and your exam closely.

A word about grading

The best way to be sure your work is scored equitably is get a problem right—there's no chance of being given the wrong amount of partial credit. We try very hard to make equivalent deductions for equivalent errors but it's very hard, especially with a large class. In some cases specific deductions and/or extra credit awards are cited below. If a mark on your exam differs from one of those you should definitely let me know. If you find that you got a lower mark than a colleague for a similar error, let me know. In some cases it might be that your friend got lucky. If so, I won't change your friend's mark but I also won't compound the error by making the same wrong deduction for you.

If you believe that an answer for one of the code-centric questions is correct then before coming to see me you should get on a machine and see if it actually works. If it does, come and see me. If it doesn't work then figure out the smallest change that makes it work. If you still think the deduction is inequitable then come and see me.

Please double-check that the scores written on each problem match the scores on the cover sheet.

As the syllabus states, you're encouraged to contest any score that you don't think is equitable.

Problem 1: (5 points; average 4.61, median 5)

The instructor often says "In ML we never change anything; we only make new things." What does he mean by that?

Here are my favorite complete answers:

"Everything is immutable."—Qiyam Tung

"There are no variables in ML, only aliases to unchangeable pieces of data."—Garrett Hoxie

A common error was to have a variable-centric answer, perhaps saying simply that a `val` might hide an earlier binding of the same name. That was a 1.5 point deduction.

Problem 2: (5 points; average 4.36, median 5)

(a) Write an ML function `fun firstLast(L)` *that returns a tuple consisting of the first and last elements of the list* `L`.

```
fun last [x] = x
  | last (_::xs) = last(xs)

fun firstLast(x::xs) = (x, last(x::xs))
```

(b) What is the type of `firstLast`*?*

```
'a list -> 'a * 'a
```

Problem 3: (5 points; average 2.28, median 1.5)

Flatten one-element lists.

```
val f = map hd
```

I was surely disappointed by the results on this one. I wish more students had put down something that worked, even if it was longer than I requested. Some students even erased answers and left it blank. Remember that on my exams something is no worse than nothing—give me a chance to give you partial credit. A blank is a zero, period.

Problem 4: (6 points; average 4.67, median 6)

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.)*

```
fun eo(x::y::more) = y::(eo more)
 |  eo _ = []
```

Problem 5: (12 points; average 8.59, median 9)

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`*.*

```
fun ints_to_string [] = ""
 |  ints_to_string L =
    let
        val almost =
            concat(map (fn(s) => ", " ^ Int.toString(s)) L)
    in
        implode(List.drop(explode almost, 2))
    end
```

I ended up working through a fairly similar problem—`printN`—during the Q&A session so I tried to compensate for that by saying a lot about it in a follow-up note to the list on Monday night.

Common errors were a trailing `", "` (-3); a `tl` too few (-1); producing just the integers, no commas (-5); and doing little more than `map Int.toString` (-8).

Problem 6: (15 points; average 13.03, median 15)

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.*

```
fun its' [] = "<empty>"
 |  its' L = ints_to_string L;

fun show_lists(L) =
    print(concat(map
         (fn(label, IL) => label ^ ": " ^ (its' IL) ^ "\n") L))
```

Many solutions for both problem 5 and problem 6 used folding instead of a `map` and `concat` (or folding with `op^`). I see that as good news and bad news: I think that shows great understanding of folding but always seek the simplest solution—it has the greatest chance of being correct.

We ended up grading this problem in a somewhat qualitative fashion. For the most part we classified solutions as "A", "B", "C", "D", or "F" and awarded 15, 12.5, 10, 5, or zero points, respectively. If you see something like "A-15" on yours, that's why.

A very common error was to return `unit list` instead of `unit`. We didn't deduct for that but did award a point of extra credit if `unit` was returned. (Be sure we gave it to you if that's what you did.)

Problem 7: (10 points; average 1.83, median 0)

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]`

A fair number of students solved this problem or got close enough to make me happy. Mr. Bressel's solution was surely the best. I'd like you to think more about this problem so I'm not going to publish a solution here but if you're dying to know, see me or get to know Mr. Stout, Mr. D. Nguyen, Mr. Ghigliotti, Mr. Shokirev, Mr. Patki, Mr. Sarando, Mr. Tung, Mr. Bressel, Mr. Sortelli, Mr. Wallis, or Mr. Nation. (alphabetical by login name)

Problem 8: (4 points; average 2.25, median 2)

(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];
```

It's valid. The type is `unit -> 'a -> unit -> 'a list`

I posted a question very similar to this to the mailing list. Remember that I think of my postings to that list as part of the course material.

There were a few exceptions but if you put down any type we usually considered that as evidence that you viewed the declaration as being valid and gave you a point for that.

(b) Write an ML function `g` *whose type is*

```
int -> (int list * int) -> (string list) -> (bool * real)
```

```
fun f 1 ([1],1) [""] = (true,1.0);
```

Problem 9: (4 points; average 2.71, median 3.25)

(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`*.*

```
val f = implode o rev o tl o rev o tl o explode;
```

(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.)*

```
string -> string
```

A few students fell victim to the bum steer about the intermediate functions. I guess they overlooked slide 162 and missed the time or two in class when I specifically said to watch out for this on an exam.

Problem 10: (16 points; average 9.85, median 12)

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.

```
exprs = []
results = []

if ARGV.length != 0
    gap = ARGV[0].to_i
else
    gap = 1
end

while line = STDIN.gets
   exprs << line.chomp
   results << (STDIN.gets || (puts "Error: short input"; exit))
end

longest = exprs.max{|a,b| a.size <=> b.size }.size

for e in exprs
    printf("%s%s\n", e.ljust(longest) + (" " * gap), results.shift)
end
```

Common deductions: Doesn't handle command line option (-2); doesn't handle "short input" case (-2); doesn't bother at all with alignment (-4); gets alignment wrong (-2). A lot of solutions aligned columns based on the longest input line instead of only considering the expressions (-2).

Problem 11: (2 points; average 1.62, mean 2)

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.

For two points of extra credit, have less than 25 characters in your solution.

```
puts readlines.reverse
```

I figured this one would be a cream puff since (1) it was the first thing we wrote in Ruby (slide 9), (2) I posed an in-class challenge to do this, and (3) I wrote the above on the Elmo when discussing (2)!

Problem 12: (3 points; average 2.02, median 3)

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`.

If `gets` produces a line then it is assigned to `line`. If not, the `return` terminates the current method.

Many students said mentioned something to the effect of "the value produced by `return` is then assigned to `line`". That resulted in a deduction of 1.5.

Problem 13: (8 points; average 5.48, median 6.25)

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`.

```
def upto_limit(a, limit)
    sum = 0
    for i in a
        sum += i
        if sum <= limit then
            yield i
        end
    end
    a
end
```

Common errors: Doesn't return `a` (-1.5); yields wrong values due to logic error (-1.5), prints instead of yielding—a fundamental misunderstanding (-2.5).

If a solution simply yielded the values in `a` we only gave it two points but at the moment seems like too big of a deduction. If you got that 2, see me and I'll change it to 3.5.

Problem 14: (5 points; average 2.85, median 4)

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.*

```
def extract(s,m,n)
    majors = s.split("/")
    majors.delete("")
    major = (majors[m-1] || return)

    minors = major.split(":")
    minors.delete("")
    minors[n-1]
end
```

A hint specified "*Assume that* `"|20|1|300|".split("|")` *produces* `["20", "1", "300"]`. With that in mind the `delete("")` calls above are not needed.

Common deductions: Using `m` and `n` instead of `m-1` and `n-1` (-0.5); not checking whether the major reference is out of bounds (-0.5).

Extra Credit Section (one point each unless otherwise indicated; average 2.86, median 2; high 8)

(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?*

The result is different. In ML it is a list containing the function `f`. In Ruby it is an array that contains the result of calling `f`.

(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.*

See slide 14 in `intro.sli.pdf`. I believe that `3+4` is a valid expression in all those languages except perhaps SIL2.

We went ahead and gave a point for just the language name. If you cited a valid expression it was worth

another half-point per language.

(3) *For one point each, write* `curry` *and* `uncurry` *in ML.*

```
fun curry f x y = f (x,y)

fun uncurry f (x,y) = f x y
```

Very few got these right but quite a few got them reversed.

(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?*

`s+""` I believe that only one student, Mr. Tung, got this one.

(5) *What element of Ruby most closely corresponds to an anonymous function in ML?*

Blocks.

(6) *Simplify this Ruby expression:* `if a[0] == nil then false else true end`

The answer I had in mind was `a[0]` but that's a little inaccurate. Mr. Kovell and Mr. Maloney did better: `!!a[0]`. (Do you see the difference?)

I should have said "Greatly simplify..." but since I didn't I accepted any expression that produced the same result but that was simpler than the above.

(7) *Cite a contribution to knowledge made by Ralph Griswold.*

I hope that somebody will take the time to write a biography about Ralph. Here are a few thoughts I've got.

At Bell Labs Ralph led development of four SNOBOL languages. I believe he said SNOBOL took a day to implement. SNOBOL2 took a week. SNOBOL3 took a month. SNOBOL4 took a year. SNOBOL4 was very popular both in industry and academia. It was written with portability in mind and coded in a macro language called SIL (SNOBOL Implementation Language). It was ported to dozens of different machines. Quite a few books were written about SNOBOL4 by both him and others. For many years it was by far the most popular unconventional language to use for non-numeric problems on mainframe computers.

A language that almost nobody's heard of is SL5—SNOBOL Language 5. Ralph led development of it here at the University of Arizona and, as the name implies, it was a follow-on in the SNOBOL line. Ralph once said, "SL5 had everything—you could even change the procedure call mechanism". But in the end the team didn't like the overall result and the language was never released. It is fading into the sands of time.

I asked Ralph what his first ideas about Icon were and he said, "I was in the hospital [I don't know for what] pondering SL5 and I kept thinking that there must be something simpler." That turned out to be Icon. Great effort was taken to make Icon small (conceptually) but powerful. I remember once that Ralph patiently listened while I rattled off a list of features that I thought would be good in Icon. He said "Go ahead, but for every one feature you add, first find one to remove." He was exaggerating, but by only a tiny amount.

Perl guru Damian Conway was asked in an interview, "What languages other than Perl do you enjoy programming in?" He replied, "I'm very partial to Icon. It's so beautifully put together, so elegantly proportioned, almost like a Renaissance painting." In the `digibarn.com` poster cited in the intro slides Icon is described as "A general purpose language known for its beauty and grace."

Ralph founded this department and served as department head for many years. It's hard to imagine how difficult it is to simultaneously and successfully teach classes, conduct original research and publish about it, write successful grant proposals, build from scratch a department that is internationally recognized, direct and coordinate the work of several graduate students, and on top of all that, keep an academic department

running smoothly, bearing in mind that a collection of professors, each with their own priorities and often-strong opinions, are not easy to keep happy.

When Ralph "retired" he quickly rose to prominence in the weaving community by virtue of his interest and pursuit of mathematical problems in weaving.

Here's a quote about Ralph from one of his graduate students who is now a professor: "As one of the founders of the Bell Labs software culture which spawned UNIX, C, and many other essential contributions to modern software, Ralph Griswold brought to his academic research not only brilliance, but also experience and a value system that demanded that research ideas be tested by fire and proven useful and usable by real users, not just good-looking diagrams in academic papers."—Clint Jeffery

When I started working for Ralph as a graduate student I discovered that my was education was just beginning. He was probably the smartest guy I've ever known and surely the wisest, at least in this field. One of the great fortunes of my life was working for him.

Things that Ralph clearly saw as good ideas over thirty years ago, such as run-time type checking, garbage collection, and a well-designed collection of datatypes are now becoming widely accepted.

(8) (Up to five points.) Offer some intelligent observations about the applicability of type deduction, or something similar, in Ruby.

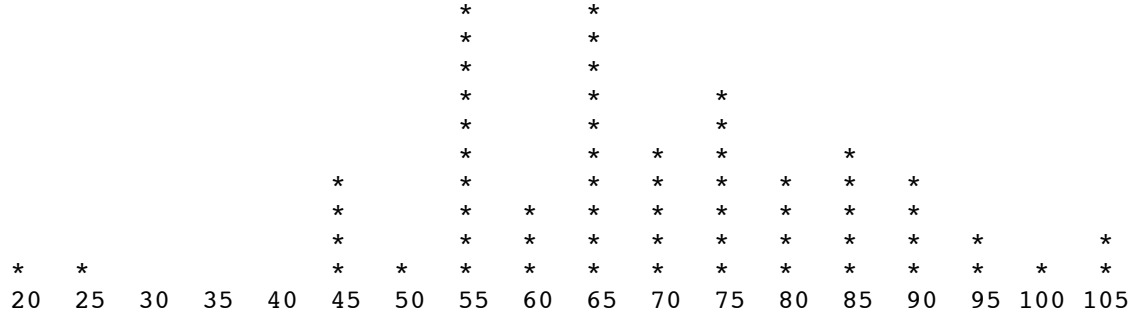
There has been quite a bit of research on various aspects of figuring out types in languages that don't use static typing, like Icon and Ruby. A fellow named Ken Walker wrote a type-inferencing Icon compiler about a decade ago for his dissertation. I believe he found that as a rule "type consistency" held in about 80% of the cases for the large body of Icon code that he tested with. You can read more here: `www.cs.arizona.edu/icon/ftp/doc/tr93_32.pdf`.

Ruby is more slippery than Icon however. For example, in Icon you can be sure that something like `x * 3.7` is always going to produce a floating point value. That's not so in Ruby. In Ruby you can't even be sure that `2+2` is `4`! However, a user who's interested in speeding up a Ruby program might be willing to sacrifice mutability for performance.

Overall results

The average on the exam was 68.9 and the median was 67.25. Here is the full set of scores and a histogram:

```
106.00, 104.00, 102.00,  96.50,  96.00,  91.50,  91.00,  90.50,  89.50,
 86.00,  85.00,  84.50,  84.50,  83.50,  82.00,  82.00,  81.00,  79.00,
 76.50,  75.50,  75.50,  75.50,  75.50,  74.50,  74.50,  71.50,  69.50,
 68.50,  67.50,  67.50,  67.00,  66.50,  66.50,  66.50,  66.00,  64.00,
 64.00,  63.50,  63.00,  62.50,  62.00,  60.50,  57.50,  57.00,  56.00,
 55.50,  55.50,  55.50,  55.50,  55.00,  53.50,  53.50,  53.50,  50.00,
 46.50,  45.25,  43.50,  42.50,  23.75,  18.50
```



I want to think about it a little bit more but I'm currently leaning against any sort of class-wide adjustment of the scores. If that were to happen I believe the maximum I'd add would be 6 points.

CSc 372, Fall 2006
Final Examination Solutions

Problem 1: (13 points; average: 10.8)

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.

```
ids = Hash.new([])
lnum = 1
while line = gets
      line.scan($id_re).each {|id| ids[id] += [lnum] unless $kwds.include? id }
      lnum += 1
end

ids.sort.each {
      |pair|
      id = pair[0]
      lines = pair[1].uniq
      printf("%s: %s\n", id, lines * ", ")
      }
```

Problem 2: (8 points; average: 6.7)

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"]]
```

Here's what I thought of right off the bat:

```
def parse_layout(s)
    r = []

    s.downcase!
    while s =~ /([mf])(-+)([mf])/
        r << [$1, $2.size, $3]
        s[$&] = ""
    end

    r
end
```

Mr. Ghigliotti and Mr. Sortelli used an approach like this:

```
def parse_layout(s)
    s.downcase.scan(/([mf])(-+)([mf])/).map {|e1,ds,e2| [e1,ds.size,e2] }
end
```

Problem 3: (2 points; average: 1.1)

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

```
class Fixnum
    def + rhs
        0
    end
end
```

Common errors were to omit `class Fixnum` and to specify more than one parameter for the + method. (Remember that `a+b` is like `a.+(b)`).

Problem 4: (4 points; average: 2.7)

(a) In a non-recursive way, implement `between(+Low, +High, -Value).`

```
between(Low, High, Value) :- numlist(Low, High, List), member(Value, List).
```

(b) In a non-recursive way, implement `numlist(+Low, +High, -Value).`

```
numlist(Low, High, List) :- findall(Value, between(Low, High, Value), List).
```

The answers for `between` were mostly correct but only about a dozen answers for `numlist` were correct.

Problem 5: (6 points; average: 4.1)

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.

```
idpfx(L,P) :- reverse(L,R), append(_, P, R), allsame(P), !.
```

Problem 6: (14 points; average: 12.9)

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.

```
show_cost(Cable) :-
      order(Cable, cable(E1, Len, E2)),
      map(E1,E1nm),
      map(E2,E2nm),
      cost(E1nm, E1Cost),
      cost(E2nm, E2Cost),
      cost(foot, FootCost),
      Cost is FootCost * Len + E1Cost + E2Cost,
      format('~w-foot ~w to ~w: $~2f~n', [Len, E1nm, E2nm, Cost]).

order(cable(E1, Len, f), cable(f, Len, E1)).
order(X, X).

map(m, male).
map(f, female).
```

Problem 7: (7 points; average: 8.1)

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(L,H1,H2) :- append(H1,H2,L), length(H1,Len), length(H2,Len).
```

When I wrote my first solution for this I checked for an even number and then used `length` to form a half-sized list. I then realized that a description of halving, "find a place to cut this so you get two equal parts", works in Prolog, too!

Problem 8: (8 points; average: 3.3)

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([W], [W]).

wseq(Ws, [W1,W2|Sorted]) :-
    select(W1, Ws, Rest),
    wseq(Rest,[W2|Sorted]),
    pair(W1,W2).

pair(A1,A2) :-
    atom_chars(A1,C1),
    atom_chars(A2,[C|_]),
    last(C1,C).
```

Problem 9: (2 points each; 16 points total; average: 12.1)

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?

> In ML it means `((f 1) 2)` — call `f` with the argument 1. Then call the function produced by `(f 1)` with the argument `2`.

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.

> I'd say that Prolog makes the naming problem worse because there's no nesting of expressions; ML makes things better thanks to partial applications and anonymous; Ruby is neutral.

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.

> When an expression is evaluated in Icon it either succeeds and produces a result or it fails and produces no result. Failure propagates to enclosing expressions and they fail in turn.
>
> Two examples of expressions that can fail are `x < y` and `s[n]`.

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

> Success or failure itself is a meaningful result that is manifested by control continuing through the goal (or not). Arithmetic expressions need to produce a result but an expression like `3 + 4` simply doesn't provide a place to specify a result. Arithmetic predicates would need to look like `add(3,4,R)` and things would get clumsy fast.

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.

A number of students said that Prolog has no type checking—type mismatches simply produce "No" but that's incorrect. Consider `X is 3 + abc`. There's no error when that code is loaded but there is an error when that goal is evaluated.

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.

I personally think this is a terrible idea but that's perhaps because I'm used to Icon, which produces a one-character string for `string[n]`.

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.

The method

```
def double x
    return x + x
end
```

"doubles" `x` by adding it to itself. This has meaning for any type that has provided a meaning for addition.

The term "duck typing" appears to have originated in the Ruby community but the idea has been around for years; routines like `double` are said to be "polymorphic".

What is meant by the term "syntactic sugar"?

A syntactic construct that is not necessary but provides a more convenient way to express something.

Problem 10: (1 point each; 4 points total; average: 2.5)

Characterize each statement below as true or false.

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

It's true that I made a practice of calling Ruby a "scripting language" but I also said that in general I don't like that term to describe languages like Icon, Perl, Python, and Ruby. The industry may well be stuck with "scripting language" by now but an alternative I prefer is "agile language". However, when you start trying to characterize agile languages you often end up with only one thing in common: dynamic typing, so maybe "dynamically typed language" is the best term.

Lisp is an interesting case: it's dynamically typed but I think its syntax is sufficiently cumbersome to make me hesitate saying it's agile.

I think the term "scripting language" should be reserved for languages that let one orchestrate a series of commands in another computational domain with little interaction between the domains and/or little knowledge of either domain in the other domain. For example, most shells support scripts that string together commands but the only communication between the two domains is via command line arguments, termination codes, and standard input/output. The shell has no knowledge of what `cat`, `ls`, `find`, etc. are doing and the programs have no knowledge of the shell—they simply take arguments, maybe handle standard input/output, and produce an exit code.

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

Just about everybody missed this one. When `write(1 to 10)` is evaluated the generator `1 to 10` produces 1,

the first value in its result sequence, and that is written out. To get all the numbers you need to put in a context where the generator is resumed until it is exhausted, like `every write(1 to 10)` or `write(1 to 10) & p()`, where `p` is a procedure that always fails.

F *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 use of three different variables allows unification with three different values but it does not require them to be different.

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

`/[a-z]x[123]/` (A one-character character class is equivalent to a single character.)

Problem 11: (18 points)

The average on the essay questions was 16.53.

Problem 12: (6 points, EXTRA CREDIT; average: 1.74)

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) :- atom_chars(S,C), subexpr(R, C, []), show(R).

subexpr(sub(Id,D)) --> id(Id), lbrack, digits(D), rbrack.
subexpr(sub(Id,D1,D2)) --> id(Id), lbrack, digits(D1), [','], digits(D2), rbrack.

show(sub(Id,D)) :- format('~w.charAt(~w)~n', [Id, D]).
show(sub(Id,D1,D2)) :- format('~w.substr(~w,~w)~n', [Id, D1, D2]).

lbrack --> ['['].
rbrack --> [']'].
```

In many answers the `format` calls were in the grammar rules. There was no deduction for that but it's not entirely correct. The problem with printing in the rules is that there may be trailing characters, like `'s[1]x'`. If so the printing is premature.

Extra Credit Section (one half-point each unless otherwise indicated; average: 4.0)

NOTE: In general, incorrect but humorous answers worked, too.

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

Lots of students said "SNOBOL5" but it was "SL5" (SNOBOL Language 5).

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

I made the briefest mention of it while looking at the Wikipedia "Multi-Paradigm" page during the last class but Oz is claimed to support eight programming paradigms. It can also be argued that due to its flexibility Lisp can support any paradigm so I accepted Lisp as an answer as well.

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

I'd say the order is Lisp, ML, Icon, Ruby, Java, Scala but Java and Ruby are very close. As long as those two were consecutive either order was acceptable for them. I counted an answer as correct if at most a single deletion yielded a valid ordering.

(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.

It wasn't very conservation-minded to allow so much space for such a short answer but here's all you need:

```
fun eq(L1,L2) = L1 = L2
```

I should have deducted a little bit for `fun eq(L1,L2) = if L1 = L2 then true else false`, but I looked the other way.

(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
```

There would be no need to see if `c` was `nil`:

```
for i in 0...len
  r += self[start+i].chr
end
```

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

James Gosling's Ph.D. dissertation concerned the algebraic manipulation of constraints.

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

What I had in mind was "Ruby meets Prolog" but other answers worked, too.

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

What I had in mind was "Java meets ML".

(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)

I was surprised by the number of good ones that turned up. I wish I had time to quote them all.

(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?

tHHGttG

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

Duke Vader

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

I believe that just about the only reading I assigned (rather than only suggested) was in the assignment 9 write-up: "There's even a Wikipedia article about pancake sorting. (Read it!)"

(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.

Zero. I prefer their waffles! I'll sometimes have the German Pancake but there was never enough time for one of those on lecture days.

Only Mr. Hoskins got this one, speculating that I was probably too busy flipping them to eat any.

(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.

```
sum = 0
while line = gets
    if line.chomp! =~ /(\d+) points( total)?\)$/ then
        sum += $1.to_i
    end
end
puts sum
```

Mr. Merrill's solution was unique: The language he chose was English!

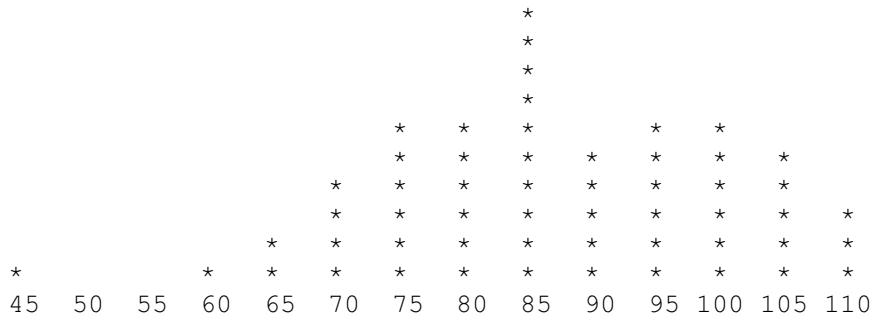
Overall statistics

The average on the exam was 86.57; the median was 85.5.

Here is the full set of scores:

```
110.00, 109.00, 108.00, 105.00, 104.50, 104.00, 103.00, 103.00, 102.00, 101.50,
99.00, 98.50, 98.00, 98.00, 97.00, 96.00, 95.00, 95.00, 94.50, 93.50, 92.00,
90.00, 90.00, 89.50, 87.50, 87.00, 87.00, 85.50, 85.50, 85.50, 84.50, 84.50,
83.50, 83.00, 82.50, 82.00, 82.00, 81.50, 81.00, 79.50, 78.50, 77.00, 76.00,
75.00, 75.00, 73.50, 73.50, 72.00, 70.50, 70.50, 69.50, 64.00, 64.00, 58.00,
46.50
```

Here is a histogram:



Solutions for
CSC 372 Haskell Mid-term Exam
Feburary 28, 2014

Problem 1: (15 points) Mean and median: 10.8, 10.5

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"
```
`[Char]` or `String`

```
(1,'x',[True])
```
`(Int, Char, [Bool])`

```
["x":[]]
```
`[[[Char]]]`

A little tricky, and gotten wrong by many, typically one set of brackets short. Think about turning `"x":[]` into `["x"]`, then `[['x']]` and then adding the outer brackets for `[[['x']]]`, to get `[[[Char]]]`.

```
head
```
```
[a] -> a
```

```
not . not . isLetter
```
```
Char -> Bool
```

```
map not
```
```
[Bool] -> [Bool]
```

```
(+1) . (0<)
```
Invalid, since `(0<)` produces a `Bool`, and `Bool`s can't be added.

```
\x -> x + 1
```
```
Int -> Int
```

```
[1,'2',"3"]
```
Invalid, due to multiple types.

```
isDigit . chr . head . (:[]) . (+3) . ord . chr
```
```
Int -> Bool
```

As I mentioned at least a couple of times in class, if a composition is valid you only need to look at the input of the rightmost function and the output of the leftmost function to know the type of the full composition.

Problem 2: (12 points) (two points each) **Mean and median: 10.7, 11**

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.*

I forgot to specify handling for empty lists for `head` and `tail` so we didn't take that case into account when grading.

```
head [] = error "empty"
head (h:_) = h


tail [] = error "empty"
tail (_:t) = t


length [] = 0
length (_:t) = 1 + length t


sum [] = 0
sum (h:t) = h + sum t


last [] = error "shortList"
last [x] = x
last (_:t) = last t


snd (_,x) = x
```

Problem 3: (10 points) Mean and median: 9.3, 10

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.

There are ***no restrictions*** *on this problem.*

A few approached this with a helper function for tracking the position and I think there were even a handful that used folding but my intention was a simple recursive solution:

```
prswap [] = []
prswap (a:b:xs) = b:a:prswap xs
```

Several students used a take 2/drop 2 approach, like this:

```
prswap [] = []
prswap list = reverse (take 2 list) ++ prswap (drop 2 list)
```

One of the general wisdoms I've learned for languages that support cons lists is that cons is always preferred over concatenation. Cons simply makes a new node but concatentation is done by making a series of cons nodes with each of the elements in the left operand. Here's the source for ++ from the Haskell Prelude:

```
(++) []     ys = ys
(++) (x:xs) ys = x : xs ++ ys
```

Something like `[1,2,3] ++ [4,5,6]` is effectively `1:2:3:[4,5,6]`

With all that in mind I'd think that the take 2, drop 2, concat version would be much slower and take far more memory but with `:set +s` in my `.ghci` I see the following.

```
% cat prswap.hs
prswap1 [] = []
prswap1 (a:b:xs) = b:a:prswap1 xs

prswap2 [] = []
prswap2 list = reverse (take 2 list) ++ prswap2 (drop 2 list)

m = [1..10000000]

run1 = length $ prswap1 m

run2 = length $ prswap2 m

% ghci prswap.hs
...
> run1
10000000
(3.87 secs, 1765864896 bytes)

% ghci prswap.hs  (Fresh run of ghci, to get a clean slate.)
...
> run2
10000000
(4.97 secs, 3325445008 bytes)
```

The cons version is faster but only by about 20%. The cons version has half as much memory throughput but I imagined it'd be a smaller fraction of the concat version. Results were similar with repeated testing.

I've clearly got a lot to learn about Haskell!

This is also a good example of a bad assumption about performance. I've seen plenty of those in my career, both by myself and others, and I've learned that nothing takes the place of running cases and looking at the numbers. But even then, pitfalls abound!

Problem 4: (10 points) Mean and median: 9.5, 10

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.

*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.*

Here's a solution written by an N. C. State University grad:

```
rotabc = map f
    where
        f 'a' = 'b'
        f 'b' = 'c'
        f 'c' = 'a'
        f c = c
```

I'd hoped that most students would immediately see this as a mapping problem. Also, my hope in requiring the solution to be written out in full detail was to make people think in terms of low repetition but I suppose that once a solution is mind there's a strong temptation to just go with it.

Lots of solutions were recursive and repetitious, like this:

```
rotabc [] = []
rotabc (c:cs)
    | c == 'a' = 'b': rotabc cs
    | c == 'b' = 'c': rotabc cs
    | c == 'c' = 'a': rotabc cs
    | otherwise = c : rotabc cs
```

Problem 5: (20 points) (ten points each) **Mean and median: 9.6, 10 for recursive; 8.3, 10 for non-recursive**

*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 _ [] = []
bigTuples n ((x,y):ts)
    | x+y > n = (x,y):rest
    | otherwise = rest
    where
        rest = bigTuples n ts

bigTuples n tuples = filter (\(x,y) -> (x+y) > n) tuples
```

Problem 6: (6 points) (5 for function, 1 for type) **Mean and median: 4.9, 6**

Write a function `fml list` that returns a 3-tuple with the first, middle and last elements of a list. 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`?*

```
fml::[a] -> (a,a,a)
fml x = (head x, middle, last x)
    where
        middle = last (take ((length x `div` 2) + 1) x)
```

Some students made this a little harder than necessary by writing a recursive function to find the middle element of a list, some with a counter and some with an `init` and `tail` combo to repeatedly take an element off both ends. I'll be honest and say that when grading we didn't take too close a look for off-by-one errors on the counter-based functions, figuring they were already self-penalizing by the amount of time they took.

Problem 7: (10 points) **Mean and median: 6.9, 9.5**

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~~ acm`*. Use* `isDigit` *to test for a digit.*

```
f elem (digs, non)
    | isDigit elem = (elem:digs, non)
    | otherwise = (digs, elem:non)
```

Here's a Thanks! and a Bug Bounty Point to Mr. Garcia for being the first to point out that I'd specified the wrong type for the folding function, ending with `elem` instead of `acm`! What a whopper!!

Problem 8: (5 points) Mean and median: 2.6, 3.5

Implement `map` *in terms of a fold.*

Your solution must look like this*:* `map f list = fold...`

```
map f list = foldr (\e acm -> f e:acm) [] list
```

Problem 9: (2 points) Mean and median: 0.6, 0.5

Without using explicit recursion, write `last` *in point-free style. Hint: Tough, and easy to get backwards! Don't worry about handling empty lists.*

Note: Wording changing at start of exam to "Write `last` in terms of a fold: `last = fold...`"

```
last = foldl1 (\_ e -> e)
```

Without that last minute wording change the problem had a trivial solution, but worth a ½ point of extra credit:

```
last = head . reverse
```

Problem 10: (10 points) (one point each unless otherwise indicated) **Mean and median: 8.0, 8**

Answer the following questions. Keep your answers brief for questions 4-10—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 points)
Ralph Griswold, in 1971.

If you see this question again it might be worth zero points if correct but -2 points if wrong! Feel free to write this information on the back of your hand before future exams in 372.

(2) Name two programming languages created ***before 1990****.*
See intro slide 20, but the first two that come to mind for me are FORTRAN and COBOL.

(3) Name two programming languages created ***after 1989****.*
Lots on slide 20 but Java and Ruby are two I'd hope you know. Haskell was forming around this time, and counted as correct for this question. Another way to solve this would be "create" two languages on the spot, perhaps named after your goldfish. They'd be vaporware, but good enough!

(4) Name one language feature you would expect to find in a language that supports imperative programming.
Looping control structures, variables.

(5) *whm often says "In Haskell we never change anything; we only make new things." What's an example of that?*
A function like reverse builds a new list from the elements of an existing list. Another: A function to remove a value from a list builds a copy of the list, not including occurrence(s) of that value.

(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?*
The ability to treat functions as values.

(7) *What is relatively unique among programming languages about the way that Haskell handles strings and lists?*
Strings are simply lists of characters.

(8) *What is a "partial application"?*
A call to a multiple-argument function that's given fewer parameters than it needs.

(9) *What is "syntactic sugar"?*
A syntactic construct that makes a language more pleasant to use but that doesn't add a new capability.

(10) *In general, what's something we can represent with a tuple that we can't represent with a list?*
A heterogenous collection of values, like a string and an integer.

Note: I originally wrote this with nine questions, allowing two points for the first question. Then while "proofreading" I noticed I was a question short, and added a question, for a total of 11 points instead of 10 points. Mr. Srivastava caught this during the exam. Let's think of the extra point as possible extra credit—we'll count the exam as 100 points for purposes of average.

Extra Credit Section (½ point each unless otherwise noted) <u>Mean and median: 1.6, 1.5</u>

(1) *What is the type of* `foldr`*?*
I like to describe it as `foldr :: (val -> acm -> acm) -> acm -> [val] -> acm`

(2) *What is the type of the composition operator?*
`(b -> c) -> (a -> b) -> a -> c`

(3) *What's meant by "lexicographic comparison"? (Hint: Don't just say "dictionary order.")*
Pair-wise comparison of a sequence of values until a pair differs, with the difference deciding the ordering.

(4) *Name a programming language created at the University of Arizona.*
Intro slide 31 names a number of them but Icon is one I'd hope you know of. A presentation on Icon that Griswold gave at N. C. State led me to come to graduate school here to work on Icon. (I got on their radar here, ultimately getting a research assistantship, by being the first person to get Icon working on IBM mainframes.)

Lots of people cited SNOBOL. Griswold continued work on SNOBOL4 here at The University of Arizona but SNOBOL through SNOBOL4 were created at Bell Labs. There was also SL5 (SNOBOL Language 5) but Griswold ultimately was not happy with it. It was never released to the public although many papers about it were published while it was being developed.

Mr. Garcia used the approach suggested on Problem 10(3) with "Grant++. I made it just now." That's good for an Original Thought!

(5) *Haskell functions like* `getLine` *and* `putStr` *return an action. What does an action represent?*
An interaction with the outside world that happens when the action is evaluated.

(6) *What is the exact type of the list* `[head, tail]`*?*
`[head, tail]` is an invalid list, since the types differ.

(7) *If you only remember one thing about the Haskell segment of 372, what will it be? (Ok to be funny!)*
If time permits I'll issue a revised version with some of the highlights of these but I'm short on time at the moment!

(8) *With Ruby in mind, cite an example of an imperative method and an example of an applicative method.*
`reverse!` and `reverse` are the first two that came to mind for me.

(9) *Write a* ***Ruby*** *method* `printN` *that prints the numbers from 1 to N on a single line, separated by commas. (1 point)*

```
def printN n
   (1..n).to_a * ","
end
```

(10) *Predict the median score on this test. (The median is the "middle" value in a range of values.)*

I was wondering if *The Wisdom of Crowds* (see Wikipedia) would apply here but not so. These were the predictions.

```
100, 100, 88, 86, 84, 84, 83.2, 83, 83, 82, 81, 81, 81, 80, 79, 79, 78, 78,
76, 76, 75, 75, 75, 75, 74, 74, 72, 72, 72, 72, 71, 71, 70, 70, 70, 68, 65,
65, 60, 50, 50, 50
```

The mean of those predicted medians is 75.2. Any prediction at all earned the ½ point.

Here are the actual scores:

```
102, 102, 101.5, 100.5, 100.5, 100, 99, 97.5, 97, 95.5, 95.5, 95, 95, 94.5,
94, 93, 92.5, 92, 92, 91, 91, 91, 90.5, 90, 90, 90, 89, 88.5, 88, 87, 86,
83.5, 83, 82.5, 81, 80.5, 79.5, 79, 78.5, 78, 78, 78, 77.5, 76.5, 76.5, 76,
75, 74.5, 73.5, 69.5, 68, 66, 64.5, 59, 58, 52, 49, 45, 29.5
```

Simple statistics:

```
n = 59
mean = 82.7542
median = 87
```

For the record, here are the changes/updates accumulated during the exam and shown on the screens:

```
Exam done: 10:52

Prob 2: head, tail: assume not empty

Prob 6 (fml): Assume odd number of elements

Prob 7: foldr is elem -> acm -> acm (not elem!!)

Problem 9 change:
Write last in terms of a fold, just like problem 8, but for last

Problem 10 (8) (9) Ok to just cite an example.

New EC: Answer Problem 9 in a different way, just following the
instructions as written, which make it trivial! :(
```

Solutions for
CSC 372 Ruby Mid-term Exam
April 11, 2014

Problem 1: (21 points) Mean and median: 17.6, 19

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. ...*

The solution I wrote when timing myself was this:

```
if ARGV.length != 1 || (not ARGV[0] =~ /^-\d+$/)
    puts "Usage: tail -N"
    exit 1
end

n = -ARGV[0].to_i

lines = []
while line = STDIN.gets
    lines << line
end

if n > lines.size
    n = lines.size
end

(-n).upto(-1) { |i| puts lines[i] }
```

With some time to think and some good ideas from student solutions, I offer this simpler version:

```
if ARGV.length != 1 || (not ARGV[0] =~ /^-\d+$/)
    puts "Usage: tail -N"
    exit 1
end

n = -ARGV[0].to_i

lines = STDIN.readlines

n = lines.size if n > lines.size

puts lines[-n..-1]
```

Problem 2: (17 points) Mean and median: 13.5, 15.5

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.

```
def load_facts(file_name)
    facts = Hash.new []

    f = File.open(file_name)

    while line = f.gets
        line.chomp!
        if line = ~ /^(\w+)\((.*)\)\.$/
            facts[$1] += [$2.split(",")]
        end
    end

    f.close
    facts
end
```

The combination of initializing the hash facts with `Hash.new []` and using `+=` allows avoiding a check to see whether `facts[$1]` already exists. (Using `<<` instead of `+=` would break it. Try it and be the first to explain why on Piazza!)

Because the input can be assumed to be valid, only processing lines that start with `/\w/` suffices to ignore empty and blank lines.

Common mistakes, shown as variations of the above, were these:

```
facts[$1] = $2.split(",")     # Doesn't build an array of arrays

facts[$1] += [$2]             # Doesn't split
```

Some students used various splits but here's the good one:

```
parts = "food(just,a,test,here)".split(/[,()]/)
```

The functor ends up in `parts[0]`; the terms are in `parts[1..-1]`.

Let's combine that with a variation of parallel assignment that we didn't study but that is similar to the mechanism for handling varying numbers of arguments:

```
>> head, *tail = "food(just,a,test,here)".split(/[,()]/)

>> head   => "food"
>> tail   => ["just", "a", "test", "here"]
```

Problem 3: (15 points) Mean and median: 9.3, 10.5

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.

My draft solution is below but Dennis noticed a bug in it. Cover the portion of the page the follows the code and play exam grader for a minute to see if you see it!

```
def print_preds h
    for k in h.keys.sort
        lens = h[k].collect {|a| a.size }
        for n in (lens.min)..(lens.max)
            puts "#{k}/#{n}"
        end
    end
    nil
end
```

The problem is that it assumes a contiguous range of "arities". For example, given `x(a).` and `x(a,b,c).`, it'll output `x/1`, `x/2` (oops!), and `x/3`. That was true with the sample data I provided but not true in general.

If had recognized the potential of noncontiguous arities, I would have mentioned `Array#uniq`, which is similar to UNIX's `uniq` command, discarding non-unique values:

```
>> [3,1,2,3,1,3].uniq    => [3, 1, 2]
```

`uniq` enables this concise solution:

```
def print_preds h
    for k in h.keys.sort
        for n in lens = h[k].collect {|a| a.size }.uniq.sort
            puts "#{k}/#{n}"
        end
    end
    nil
end
```

Without `uniq` you need some other way to compute the set of arities. The first thing that comes to mind based on what you've seen is using the | operator to form the union of two arrays:

```
>> [3,1]|[1]|[2] => [3, 1, 2]
```

The keys of a hash also constitute a set. A couple of students took that observation further and used the whole predicate indicator, like "`food/1`", as the key! Here's a solution based on that idea:

```
def print_preds h
    indicators = {}
    h.each { |k, arrays|
        arrays.each { |e| indicators["#{k}/#{e.size}"] = 1 }
        }
    puts indicators.keys.sort
    nil
end
```

There's a slight bug with that because, for example, "`t/10`" collates before "`t/2`". With these facts,

```
t(x).
t(x,x).
t(x,x,x,x,x,x,x,x,x,x).
```

you'll this output:

```
t/1
t/10
t/2
```

Also, I never mentioned it in lecture but Ruby does have a `Set` class:

```
>> require 'set'          => true

>> [3,1,3,4,2,1].to_set   => #<Set: {3, 1, 4, 2}>
```

I apologize for the unintended difficulty due to my error when writing my own solution, but everybody was in the same boat.

Problem 4: (5 points) Mean and median: 4.0, 4.5

Write a method `multstring(spec, s)` that behaves like this:

```
>> multstring("3,1,5", "x")
=> "xxx,x,xxxxx"
```

Solution:

```
def multstring(spec,s)
    spec.split(",").map {|e| s*e.to_i} * ","
end
```

Problem 5: (4 points) **Mean and median: 2.4, 2.5**

Write a method `addrev` *such that after calling* `addrev`*, the unary minus operator applicatively reverses a string.*

```
def addrev
    eval("class String; def -@; self.reverse; end; end")
end
```

I didn't expect students to know it but that class declaration must be wrapped in an `eval(...)`. The error "`class definition in method body`" is produced by this version:

```
def addrev
    class String
          def -@; self.reverse; end; end
end
```

Problem 6: (11 points) (One point each unless otherwise indicated) **Mean and median: 7.2, 8.5**

(a) *Write a regular expression that is equivalent to* `/x+/` *but that doesn't use the* `+` *operator.*

```
/xx*/
```

(b) *Write a regular expression that is equivlent to* `/ax|bx|cx/` *but that doesn't use the* `|` *operator.*

```
/[abc]x/
```

(c) *Describe in English the strings matched by* `/a*b+c$/`

Strings containing zero or more "`a`"s immediately followed by one or more "`b`"s immediately followed by a "`c`" at the end of the string.

(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).

```
/^0[bB][01]+$/
```

(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`").

```
/^[1-9][0-9]+(, ?[1-9][0-9]+)*$/
```

Problem 7: (7 points) Mean and median: 4.7, 6

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.

```
def take_while a
    r = []
    a.each { |e|
        if yield e
            r << e
        else
            return r
        end
        }
    return r
end
```

Problem 8: (12 points) Mean and median: 7.9, 8.5

(a) Who invented Ruby? (1 point)

"Matz" is the answer I was hoping for, but "some Japanese guy" was counted as correct, too. Even "Some whack-a-doo! Monkey patching?!? really?" got a 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)

Quite a few students pointed out that you'd need to do length-1 to compute the position of the last element but I was looking for something a little deeper.

I wish I'd stipulated "without overloading" but since I didn't, pointing out that you'd need overloaded methods was counted as correct. However, `char last(String s)` is easy enough but how about the array case? `int last(int a[])` only covers `int`s, of course. Can it be done with generics?

(c) What's the effect of adding "`include Enumerable`" to a class definition? (2 points)

Assuming that the class implements `each`, including `Enumerable` gives your class all the methods in `Enumerable`, as you saw with `XString`.

(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)

In both Haskell and Ruby `if-then-else` is an expression, producing a value. Haskell requires an `else` clause but Ruby does not.

(e) A Java newcomer to Ruby might think that `private` and `attr_reader` are keywords but they aren't. What are they? (2 points)

Methods

(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.

In my mind the essential difference is that with static typing you can detect a particular class of type errors without executing the code but with dynamic typing you cannot.

A number of students had a successful answer that included `/(interprete|compile)[rd]?/` and that was fine but I suggested not using that because I feared answers like "Dynamically typed languages must be interpreted and statically typed languages must be compiled.", which is completely wrong!

Problem 9: (8 points) Mean and median: 4.8, 6

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.

```
class Ring < Shape
    def initialize(label, r1, r2)
        super(label)
        if r1 < r2
            r1, r2 = r1, r2
        end
        @outer, @inner = r1, r2
    end

    attr_reader :inner, :outer
```

```
        def area
            @outer**2*Math::PI - @inner**2*Math::PI
        end

        def to_s
            "Ring #{label} (#{@inner}-#{@outer})"
        end
    end
```

Extra Credit Section (½ point each unless otherwise noted) <u>Mean and median: 3.0, 3.25</u>

(a) What does whm consider to be the all-time greatest Original Thought ever put forth by one of his 372 students?

Joe Astier, in my Fall 1996 class, likened a Prolog predicate to a motor: If you apply current to a motor, it turns the rotor; if you turn the rotor, current is produced.

Technical note: My understanding is that it must be a DC motor with a magnet.

(b) Based on the Prolog we've covered, is Prolog statically typed or dynamically typed? (And why?)

Dynamically typed

(c) What thing in Ruby is closest to a Prolog atom?

Symbols

(d) Which is the oldest of Java, Haskell, and Ruby?

Haskell

(e) The term "mixin" came from a business that sold what to college students?

The business was Steve's Ice Cream but I imagine that Steve sold other stuff, too.

(f) Regular expressions are Type <u>4</u> languages in the <u>Chomsky</u> hierarchy of languages.

(g) Predict the median score on this exam. (The median is the "middle" value in a range of values.)

Here are the predictions: (mean = 74.4, mode = 75, median = 75)

```
50, 50, 60, 65, 65, 67, 67, 70, 70, 70, 70, 70, 72, 72, 72, 72.71,
74, 74, 75, 75, 75, 75, 75, 75, 78, 78, 79, 79, 80, 80, 81, 82,
82, 82, 83, 83, 84, 84, 84.726953281, 85, 85
```

Here are the actual scores:

```
104, 103, 100.5, 100.5, 99, 98.5, 96, 96, 94.5, 93, 91.5, 91.5,
90, 90, 89, 89, 88.5, 88, 88, 86, 86, 85.5, 84, 83.5, 82.5, 82,
81.5, 79, 79, 78.5, 78.5, 78.5, 78.5, 78, 78, 78, 77, 75, 73.5,
70.5, 69.5, 69.5, 69, 68, 66.5, 65.5, 64.5, 63, 62.5, 60.5, 57,
53, 48.5, 44, 38.5, 38, 31.5, 28.5, 15.5
```

Simple statistics:

```
N = 59
mean = 75.8729
median = 78.5
```

If I had more time I'd compile the interesting answers from the following but alas I don't have that time to spare at the moment. If time permits, I'll issue an update.

One man's trash is another man's treasure so as you'd expect, what seemed stupidest to some, like monkey patching, seemed best to others.

(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)

```
def attr_reader *attrs
    attrs.each { |attr|
        eval("def #{attr}; @#{attr}; end")
        }
end
```

The real `attr_reader` is a method of `Module` but the above works as-is.

Solutions for
CSC 372 Final Exam
Wednesday, May 14, 2014

Problem 1: (6 points)

Mean and median: 5.1, 6

Cite three things about programming languages you learned by watching your classmates' video projects. Each of the three should be about a different language.

I tallied the languages cited in the responses and came up with these numbers:

Java	34
PHP	19
Lua	17
Python	15
Ruby	12
Lisp	8
C#	7
MATLAB	5
Haskell	5
C++	4
C	4
vimscript	3
R	3
Dart	3
JavaScript	2

The relatively high interest in Lisp reflected above reminds me that I should probably be whipped for not spending at least a little time on Lisp in this course. I did do a sidebar slide on Lisp in the Haskell set (#127) but ended up skipping it due to time. If you want to see a little bit of Lisp in the context of Emacs Lisp, see http://www.cs.arizona.edu/~whm/352/elisp.sli.pdf.

Problem 2: (6 points)

Mean and median: 5.1, 5.5

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.

```
last([X],X).
```

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

member(?Elem, ?List) specifies the relationship that Elem is a member of List.

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

length(?List, ?Len) specifies the relationship that List is Len elements in length.

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

Problem 3: (8 points)

Mean and median: 6.8, 7.5

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] .
...
```

Solution:

```
insrel([],[]).
insrel([X],[X]).
insrel([X,Y|T], [X,Sym|R]) :- which(X,Y,Sym), insrel([Y|T], R).

which(X,Y,'<') :- X < Y.
which(X,X,'=').
which(X,Y,'>') :- X > Y.
```

Problem 4: (7 points)

Mean and median: 5.7, 7

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".

```
alltails([_|T],T).
alltails([_|T],R) :- alltails(T,R).
```

Problem 5: (10 points)

Mean and median: 8.2, 9

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.
```

Solution:

```
head([H|_],H).
pinch(L, pair(First,Last)) :- head(L,First), last(L,Last).
pinch([_|T], pair(F,L)) :- append(Rest,[_],T), pinch(Rest,pair(F,L)).
```

Problem 6: (5 points)

Mean and median: 4.2, 5

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.*

```
sumvals :- findall(Val, v(Val), Vals), sumlist(Vals,Sum), retractall(v(_)),
      assert(v(Sum)).
```

The first version of my solution had the following no-facts case but just as I was about to deduct points for a solution not having it, I realized it wasn't needed!

```
sumvals :- \+v(_), assert(v(0)), !.   % Not needed!
```

Problem 7: (12 points)

Mean and median: 9.2, 11.5

`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.
```

Solution:

```
path(p(X,Y),p(X,Y),_,[]).

path(p(X,Y), p(DX,DY), Moves, [m(MX,MY)|MoreMoves]) :-
    select(m(MX,MY), Moves, Remaining),
    NX is X + MX, NY is Y + MY,
    path(p(NX,NY), p(DX,DY), Remaining, MoreMoves).
```

Problem 8: (10 points)

Mean and median: 6.4, 7.5

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`*.*

```
ptime(Spec,Mins) :- atom_chars(Spec,Chars), spec(Mins,Chars,[]).

spec(Mins) --> int(Hours), ['h'], { Mins is Hours * 60 }.
spec(Mins) --> int(Mins), ['m'].
spec(Mins) --> int(H), [':'], int(M), {Mins is H * 60 + M}.
```

Problem 9: (7 points)

Mean and median: 5.7, 6

*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
```

For the type, it was fine to have something like this:

```
ckconn :: Char -> [(Char,Int,Char)] -> Char -> Bool
```

but the type that Haskell infers is this:

```
ckconn :: Eq a => a -> [(a, t, a)] -> a -> Bool
```

Here's the executable code:

```
ckconn _ [] _ = False

ckconn from [(left,_,right)] to =
      from /= left && to /= right

ckconn from ((left,_,right):t) to =
      from /= left && ckconn right t to
```

Problem 10: (7 points)

Mean and median: 5.3, 6

*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
```

Solution:

```
connlen :: Char -> [(Char,Int,Char)] -> Char ->  IO ()
connlen from cables to = putStr result
    where
      result =
        if ckconn from cables to then
            (show $ sum $ map ( \(_,len,_) -> len) cables)
                  ++ " feet\n"
        else
            "nope\n"
```

Problem 11: (14 points)

Mean and median: 11.9, 14

*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.

Here's a "plug and chug" solution:

```
if ARGV.size != 1 || (not ARGV[0] =~ /^-\d$/)
    puts "Oops!"
    exit 1
end

n = -ARGV[0].to_i

blocks = []
while true do
    block = []
    1.upto(n) {
        line = STDIN.gets
        if !line then
            blocks.shuffle.each { |block|
                block.each {|line| puts line }
                }
            exit
        end
        block << line
        }

    blocks << block
end
```

Those nested `each` blocks for output can be replaced with just this:

```
puts blocks.shuffle
```

Note that I "punt" on getting out of the `1.upto(n)` block by printing and exiting in the middle of the block when end of file is reached.

If you lean on the library a little more you can come up with this:

```
if ARGV.size != 1 || (not ARGV[0] =~ /^-\d$/)
    puts "Oops!"
    exit 1
end

n = -ARGV[0].to_i

blocks = []

lines = STDIN.readlines.each_slice(n) { |slice| blocks << slice }

puts blocks.shuffle
```

My son pointed out that PHP has `array_chunk`. That suggests a one-liner like this,

```
puts STDIN.readlines.chunks(n).shuffle
```

but I can't find anything like `chunks` in Ruby, which PHP programmers know as `array_chunk`. However, we can correct that omission with this:

```
class Array
    def chunks(n)
        result = []
        self.each_slice(n) { |slice| result << slice }
        result
    end
end
```

I first encountered "shuffile" years ago as an Icon program. At the time I assumed that Ralph Griswold came up with that clever name but maybe it was somebody else. There's a `shuf` Linux command, too. It has the useful behavior of being able to shuffle command-line arguments as well but its `-i` and `-n` options make me wonder if the author knew of `seq` and `head`.

Problem 12: (8 points, one point each)

Mean and median: 5.7, 6

(a) Who founded UA's CS department, and in what year?

Ralph Griswold, in 1971. I hope the CS majors will always remember who got this program going.

Ralph once said that on his first day here, he arrived to find a number of students waiting in line outside his office, for advising. Once in his office, he found that he had a desk but no chair.

Ralph said he took the position here with the specific condition that he would not have to serve as department head but when no other suitable candidate was found, he took on that role.

A random memory about Ralph: A tradition started by his graduate students was showing up at his house on Halloween night, and inviting themselves in for a chat. Those Halloween visits by students and former students continued until his death, in 2006.

(b) Why are Prolog warnings about singleton variables worth paying attention to?

A singleton warning often indicates a misspelled variable name. In other cases it might indicate that a value is being computed but never used, so why compute it.

(c) What's the fundamental difference between predicates like `member/2` and `append/3` in Prolog and their superficial analogs in other languages?

Prolog's `member` predicate can be used to both test for membership in a list, generate all elements in a list and more.

(d) Draw the box for Prolog's four-port model and label the ports.

I'm lazy! See slide 55.

(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?

Many problems have a natural recursive formulation of operating on the first element of the list and combining that with the result of the function on the rest of the list.

(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.

Grammar rules are a great example of syntactic sugar. Any grammar rule can be brainlessly rewritten by adding

"In" and "Out" arguments to each non-terminal, replacing terminals with list unifications, and removing curly braces around ordinary goals.

(g) *Write a simple **Haskell function** and show an example of using a partial application of that function.*

```
> let add x y = x + y
> map (add 5) [1,2,3]
[6,7,8]
```

(h) *What's a very good reason that `ckconn` above uses tuples instead of lists to represent the cables?*

Those Prolog lists are heterogeneous.

Extra Credit Section (½ point each unless otherwise noted)

Mean and median: 2.3, 2.5

(a) *What movie inspired the "Original Thought" bonus?*

Broadcast News

(b) *whm came to graduate school here at UA specifically to join a research team developing a programming language. What was the language?*

Icon

(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."

Dr. Proebsting

(d) *The Prolog 1000 is a compilation of applications written in Prolog and related languages. What's a little odd about it?*

There are less than 1000 applications on the list.

(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?*

Lisp

(f) *Cite a significant contribution to computer science made by Grady Booch.*

He was one of the "Three Amigos"—along with James Rumbaugh and Ivar Jacobson—who created UML.

(g) *What's a language that has/had an "arithmetic if", one form of which looks like this:*

```
     IF (I-J) 100, 110, 120
```

FORTRAN. Control branches to the statement labeled `100`, `110`, or `120` respectively, depending on whether the result of `I-J` is negative, zero, or positive.

(h) *In terms of creators, what do `vim` and Java have in common?*

Bill Joy, later a founder of Sun Microsystems, wrote `vi`, which of course inspired `vim`. Joy was also involved with Java early on.

Several students found success with something like "They were both created by programmers."

(i) *Market research has determined that the course title Comparative Programming Languages is dry and unappealing. Think up a new name for 372. It needn't be less dry but should be funny!*

The responses are below, in random order. It seems like not everybody got the "It needn't be less dry but should be funny!" part, but I suppose good humor is where you find it. I also posted the list on the board outside the 733 corridor. If you're in the building, go by and vote for your favorite.

Learn 3+ Languages in Too Short of Time
Brain Buster: The Class
So Much Recursion
Get Confused by Small Differences 101
372: Fiesta of Programming
Resume Booster
Why Ruby is So Much Better Than Haskell [Ed. note: That wasn't my intention!]
The Many Languages of Pancakes
Dr. Strangelanguage or: How I Learned to Stop Worrying and and Love Prolog
Super Comparative Programming Languages
Actually Learn Programming
372: The Wrath of Cons
More, Harder, and Funner Homework Than You Could Hope For
Study of Mind-Warping
Trying to Keep 3 Languages Separate in Your Head
Multiple Languages Just in One Semester
Exploration into the World of Machine Languages
Heads and Tails, Let's Rock!
Three Fun Ways to Scramble Your Brains
Multiple Language Ingestion
How to Talk To Computers
Introduction to Different Styles of Thinking About Programming General Purpose Computing Devices
An Introduction to Remaining Competitive In Your Field, Good Luck.
Bunch of Random Languages
No Language to Beat 'em All
Learn to Hate Pancakes in 3 Different Languages
372: Too Much Information You Won't Retain
Learn All the Languages!!! or Just 3 More
Completely Pleasant Languages
Hardcore 3 Programming Language Learning in 3 Months
Learn You a Haskell /[Haskell|Ruby|Prolog]/ for Great Good
Foreign Languages for Programmers
Superfast Computer Language Learning Time
A Class About Not Java
whm and the 3 bears
Haskell, Ruby, Prolog! (so they know the languages we're learning)
Any Way You Want It: 3 Languages for the Programmer on the Go
A Programmer's Nightmare
1001 Ways to Write Fast
Wow So Many Languages in One Course
Too Many Languages!
Not Java—the Class!
French 102 and Other Languages (so us B.S. students wouldn't have to take Foreign Language in place of a CS elective!)
Pancake Hell
Programming in Latin
How Many Examples Can We Fit Into One Slideshow
Programming Languages: The Good, The Bad, and Haskell
Blow Your Mind With Unorthodox Programming Languages

Functional, Objective, Logical—A Plethora of Paradigms
Programming Languages, A Love Story
Learning How to Stroke Egos: New With Haskell!
Programming Paradigms Explored (not funny, but EC!)

(j) whm would like to recycle a8's `buy.pl` *in a future semester but needs more* `dontmix` *facts with an element of humor. What's another pair he could use?*

My favorites of these were `dontmix('Godzilla','NYC')` and the classic but elegant `dontmix(oil,water)`.

(k) How many students earned the "close reading" bonus on assignment 8?

Eleven.

Answers like "A few." were counted as correct, too.

Statistics

All scores:

```
100, 100, 99.5, 98.25, 98, 97.5, 97, 97, 96.75, 96.5, 96, 95, 94.5, 94.5, 93.5,
92.75, 91.75, 91, 91, 90.5, 90.5, 89.75, 89, 88, 87.5, 87, 86.75, 86.5, 84,
82.5, 82.5, 82.5, 81, 80.5, 80, 80, 77.5, 76.5, 76.25, 76, 75.5, 74.25, 72.5,
70, 65, 62, 61, 60, 59, 59, 58.25, 57.5, 54, 44.5, 41
```

```
N = 55
mean = 81.6136
median = 86.5
```

Solutions for
CSC 372 Mid-term Exam
Thursday, March 12, 2015

A note about grading mistakes

I wish I was perfect but I'm not. I do make mistakes when grading. I encourage you to double-check my work and let me know if you find any mistakes or disagree with any of my grading decisions, including deduction amounts.

I do regrading on a problem by problem basis; a regrade request for a single problem does not trigger an audit of the full exam.

There's no deadline for reporting grading mistakes aside from the filing deadline for final grades.

Unless there are exceptional circumstances I require regrading requests to be in person, not via email or IM.

Before asking me to revisit any problem involving code, first type in your code and see what it does.

I hate to have to mention it but note that all completed exams were scanned into a PDF, making it easy to catch post-grading alteration/addition of answers. Don't be tempted to add a couple of characters to get a point or two back!

Problem 1: (15 points) mean = 10.18, median = 11.00

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'
```
Char

```
("len", "", "")
```
([Char], [Char], [Char])

```
(:)
```
```
a -> [a] -> [a]
```

```
[(1,'a'),('b',2)]
```
Invalid. The two tuples have different types.

```
map length
```
```
[[a]] -> [Int]
```

```
(+1) . (0<)
```
The composition is not valid. `(0<)` produces a Bool but `(+1)` requires a number.

```
head
```
```
[a] -> a
```

```
(True, (False, "x"))
```
```
(Bool, (Bool, [Char]))
```

```
length . (\x -> [(x,x)]) . head . init . tail
```
Hint: the composition is valid.
```
[a] -> Int
```
If you wrote `... -> ... -> ... -> ... -> ...`, see slide 252.

```
[1..] < [2..]        (Remember: I want the type!)
      Bool
```

Problem 2: (12 points) (two points each) **mean = 9.92, median = 10.50**

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

```
head (h:_) = h
```

tail

```
tail (_:t) = t
```

length

```
length []= 0
length (_:t) = 1 + length t
```

max *(Hint: `max 2 3` is 3)*

```
max x y
  | x > y = x
  | otherwise = x
```

maximum *(`maximum [5,2,3]` is 5. Ok to abbreviate as `mm`.)*

```
mm [x] = x
mm (x1:x2:xs)
  | x1 > x2 = mm (x1:xs)
  | otherwise = mm (x2:xs)
```

Another way. The explicit type specification avoids a "monomorphism restriction" error.

```
mm::Ord a => [a] -> a
mm = foldr1 max
```

A third: `mm [x] = x; mm (x:xs) = max x $ mm xs`

filter *(`filter odd [1,2,3]` is `[1,3]`. Ok to abbreviate as `flt`.)*

```
filter _ [] = []
filter f (x:xs)
    | f x = x : filter f xs
    | otherwise = filter f xs
```

Problem 3: (14 points) (7 points each) **recursive: mean = 6.05, median = 6.00; non-recursive: mean = 4.64, median = 6.50**

*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"
```

Solutions:

```
tupruns [] = ""
tupruns ((n,c):ts) = replicate n c ++ tupruns ts

tupruns = concat . map (uncurry replicate)
```

A number of non-recursive solutions looked like this:

```
tupruns list = foldr (\(n,ch) acm -> r n ch ++ acm) "" list
```

It'd be in point-free style if only `list` weren't present on both sides.

A common mistake was mapping or folding with a non-function value, like this:

```
... concat . map (replicate x y) ...
```

Problem 4: (3 points) <u>mean = 2.32, median = 3.00</u>

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]
```

Solution:

```
flattups = foldr (\(a,b,c) acm -> a:b:c:acm) []
```

A number of students had not wrong but long answers due to writing three helper functions to extract the first, middle, and last element from three tuples.

Problem 5: (7 points) <u>mean = 4.18, median = 5.50</u>

The following function removes <u>consecutive duplicates</u> 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!"
```

<u>For this problem you are to write a folding function `ff` that will work with `dropConsecDups` as shown above.</u>

```
ff val [] = [val]
ff val all@(a:_)
    | val == a = all
    | otherwise = val:all
```

Problem 6: (6 points) mean = 4.44, median = 5.50

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.

```
avglen bytes = (show $ totalChars / numLines) ++ "\n"
    where
        lineLengths = map length $ lines bytes
        totalChars = fromIntegral $ sum lineLengths
        numLines = fromIntegral $ length lineLengths
```

The version above breaks the input into a list of `lines`. I wondered how much faster it would be to just count newlines by filtering on them. The version below takes that approach.

```
avglen bytes = (show $ totalChars / numLines) ++ "\n"
   where
      numLines = fromIntegral $ length $ filter (=='\n') bytes
      totalChars = (fromIntegral $ length bytes) - numLines
```

I then tried simply counting newlines:

```
avglen bytes = (show $ totalChars / numLines) ++ "\n"
    where
        numLines = fromIntegral $ count '\n' bytes
        totalChars = (fromIntegral $ length bytes) - numLines
        count c s = foldr
          (\elem acm -> if elem == c then acm + 1 else acm) 0 s
```

The times for the three surprised me:

Use `lines` to break `bytes` into lines:	0.817s
`length $ filter (=='\n') bytes`:	0.644s
Count with `foldr`:	1.160s

These solutions handle the integer division properly but that was not required of student solutions.

Very few students remembered to append a newline to the output. Those who did got a ¼-point bonus.

I wish I'd mentioned in the write-up that the value `bytes` produced by `readFile` is a string like `"xxx\ny\nzz\n"` but some assumed it was a list of lines. I also wish I'd reminded about the `lines` function in the Prelude; some students wrote their own version, and that wasn't my intention.

Problem 7: (7 points) mean = 6.38, median = 6.50

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. ... 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`.

```
sum = n = 0
while line = gets
    line.chomp!
    sum += line.size
    n += 1
end

printf("%.1f\n", sum / n.to_f)
```

Several students approached the problem by building an array with line lengths and then summing the lengths. That seems like a Haskell solution written in Ruby!

The only common mistake was to not `chomp` or otherwise account for the line terminator(s) that `gets` leaves in place.

Problem 8: (7 points) mean = 5.51, median = 6.00

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.

```
n = -ARGV[0].to_i

while line = STDIN.gets
    line.chomp!
    puts line.split[0,n] * " "
end
```

A common assumption was an analog to Haskell's `concat` to join an array of strings together into a string but `concat` wouldn't put spaces between them. Something like `array * sep` (as shown above), or a loop was needed. Assuming something like Haskell's `unwords` was reasonable, if explicitly stated.

A common implied assumption was that `puts array[0,n]` would output blank-separated words but it actually outputs one array element per line.

A number of students assumed a single-digit N even though there is a `-100` example.

Lots of students wrote mysterious code to process multiple arguments and/or error checking code. None of that resulted in deductions but I imagine it ate up a lot of time. Avoid that on the final by taking advantage of what can be assumed.

Problem 9: (6 points) mean = 4.07, median = 5.00

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"]
```

Here was my first solution. I forgot the return on the first run but it worked on the second run.

```
def chunkstr(s,n)
    r = []
    s = s.dup
    while s.size >= n
        r << s[0,n]
        s[0,n] = ""
    end
    r
end
```

I then wrote a more numeric solution:

```
def chunkstr(s,n)
    r = []
    pos = 0
    while pos + n <= s.size
        r << s[pos,n]
        pos += n
    end
    r
end
```

It took three runs to get right: (1) forgot `pos += n`, (2) used `s[0,n]`, (3) used `s.size + 1`. I might have also forgotten to return `r` on the first run if I'd started from scratch instead of hacking up my first solution.

Lots of student solutions did something like

```
a = []; apos = 0
...
a[apos] = str
apos += 1
```

but `a << str` or `a.push(str)` is simpler.

Problem 10: (6 points) (one point each unless otherwise indicated) **mean = 2.70, median = 2.50**

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
```

It seems that only TODO students know that the type operator `->` is right associative.

```
(Int -> Int) -> (Int -> Int)
```

Expect to see a question like this on the final.

(2) Fully parenthesize the following expression.

```
f  g  a  +  x  y  z  +  f1  (a, b)  c
```

Quite a few students got tripped up on this one but only two rules come into play: (1) two expressions side-by-side is function call. (2) function call has higher precedence than any operator.

```
((f  g) a) + ((x y) z) + ((f1  (a, b))  c)
```

Expect to see a question like this on the final!

(3) Rewrite the following expression to use as few parentheses as possible.

```
len ((map f) (head (lines bs)))
```

Solution: `len $ map f $ head $ lines bs`

(4) Describe in English the data structure matched by this pattern: `[t:h]`
A list containing a single non-empty list.

A number of students got tripped by a head named `t` and a tail named `h` but those names mean no more to Haskell than do `ping` and `pong`!

(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.)*

The use of the `==` operator in `empty2` produces the added requirement that the type `a` must be in the `Eq` type class. `empty1 [length]` works but `empty2 [length]` produces a type error because function types aren't members of the `Eq` type class.

I'd initially planned each of the two questions to be worth a point each but few students got the second question right. I decided to score the first question as two points and the second question as a one-point bonus question.

An answer that was as little as "`==`" earned full credit on this problem.

Problem 11: (7 points) (one point each unless otherwise indicated) **mean = 3.07, median = 3.00**

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
```

Solution:

```
def itr x
    yield x * 2
end
```

(2) *In the Ruby code* `$x = N * 5`*, we know that* `$x` *is a* global variable *and* `N` *is a* constant.

(3) *Given* `h = {}`*, write an expression such that after it is evaluated,* `h["x"]` *is* `10`*.*

```
h["x"] = 10
```

(4) *What's a big mistake in the following statement?*
"The `if-else`*,* `while`*, and* `for` *statements are examples of Ruby control structures."*

`if-else`, `while`, and `for` are expressions—they all produce a value!

(5) *A method named* `show_match` *is used extensively on the regular expression slides. Briefly, what does it do?*

See slide 192!

`show_match` is in my `.irbrc`. To get a copy of my `.irbrc`, do the following `cp`, assuming you're in your `a4` directory:

```
$ cp a4/.irbrc ~
```

After copying it, browse it with `less ~/.irbrc` and see if you learn anything new.

(6) *What are the minimum and maximum lengths of strings that can be matched by the following regular expression? (Be careful!)*

```
[Aa]..[Zz]0x9?
```

Six or seven characters are matched: 'A' or 'a', then any two characters, then 'Z' or 'z', '0', 'x', and maybe a '9'.

(7) Write a Ruby method that exemplifies duck typing. There's no need for any explanation!

I like the `double` example on slide 126 but most any method that invokes a method on an argument or applies an operator to an argument provides an example of duck typing.

Problem 12: (10 points) (one point each unless otherwise indicated) **mean = 6.48, median = 7.00**

Answer the following general questions. Keep your answers brief—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?

Ralph Griswold, in 1971. There was no partial credit on this one—you had get the year right, too. Here are two ways to remember 1971: (1) The decimal value for ASCII 'G' is 71. (2) There are 7+1 letters in "Griswold". Note that it's Griswold, not Griswald. (Remember the 'O's in SNOBOL.)

Good news: you'll have one more chance to get this right!

(2) Name a language that was created before Ruby. Name a language that was created after Ruby.

Popular pre-Ruby examples were FORTRAN, COBOL, C, and Icon. Common post-Ruby examples were Java and Swift. Python was a common wrong answer for a language that came after Ruby.

Two students created a language on the spot.

(3) whm believes the acronym LHtLaL appears nowhere on the web except in his slides. What does it stand for?

Learn(ing) How to Learn a Language

*(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.*

See Haskell slide 3 and following.

(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?

The ability to pass a function to a function as a parameter.

(6) What's an important difference between an imperative method and an applicative method? (Hint: it doesn't concern the method name!)

An imperative method changes state in an object. An applicative method produces a new object.

(7) whm often says "In Haskell we never change anything; we only make new things." What's an example of that?

To reverse a list we'd make a new list.

(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)

Type, value, side-effects

(9) *Briefly define the term "programming language". (1.5 points)*

Here are some of the answers I especially liked:

"A language that is used to communicate with a computer."—J. Naranjo

"A group of syntactic and semantic rules to tell a computer how to process input."—R. Storck

"A grammar and vocabulary for providing instruction in math and logic."—B. Whitely

"A collection of syntax and words which can be read by the machine."—J. Wolfe

"A compilation of syntax and semantics that supports a variety of computations."—A. Zarbock

Extra Credit Section (½ point each unless otherwise noted) <u>mean = 2.03, median = 2.00</u>

(1) *Predict the median score on this test. (The median is the "middle" value in a range of values.)*

All predictions:

```
90, 88, 86, 85, 84, 84, 84, 83, 83, 83, 83, 83, 83, 82, 82, 82, 82, 82, 82, 82,
82, 82, 82, 81, 81, 81, 81, 81, 80, 80, 80, 80, 80, 80, 80, 80, 80, 80, 78, 78,
78, 77, 77, 76, 76, 76, 76, 75, 75, 75, 74, 72, 72, 70, 70, 67, 65, 64

n = 58
mean = 79.22
median = 80
```

(2) *Add a dunsel to the following function definition*

```
f x = take 3 x ++ drop 3 x
```

Two common ones were `++ []` and `++ take 0 x`

(3) *Why was `\` chosen for use in Haskell's lamba abstraction syntax?*

It resembles λ.

(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."

Dr. Proebsting

(5) *What's the basic idea of whm's so-called "O(1) navigation"?*

Being able to hit a page with a short, fixed set of keystrokes.

(6) *What is the exact type of the list `[head, tail]`?*

The expressions is invalid—`head` and `tail` have different types.

(7) *What's interesting about the ASCII code sequence from 60 through 62?*

They form the "spaceship" operator: `<=>`

(8) What Ralph say when a young and eager graduate student put forth a list of potential new features for Icon?

Roughly quoting, "Add all the features you want but for every feature you want to add, first find one to remove."

(9) If you only remember one thing about the Haskell segment of 372, what will it be? (Ok to be funny!)

"The compiler is always right but it will never tell you why."—B. Streeter

"Jeremiah going over my head in six words."—S. Stephens

"Haskell is really spelled 'Haskhell'."—J. Scarim

"Higher-order functions are the bees knees."—C. Macdonald

"Pancakes"—A. Rane

"Typing errors, typing errors everywhere."—J. Naranjo

"Watch the types!!!"—J. Knott

"Pancakes are evil!"—K. Enami

"It's a (fun)ctional language"—S. Desai

(10) What is Matz' full name?

Yukihiro Matsumoto

(11) Write a good extra credit question and answer it.

Lots of good ones but I'm out of time to cite them here! :(

Statistics and an adjustment

Here are all scores, in order:

```
103.75, 101.25, 97.25, 97.25, 95.75, 94.00, 92.25, 92.00, 91.00, 90.50, 89.75,
88.75, 88.25, 88.25, 88.25, 87.00, 86.25, 83.00, 82.75, 82.50, 82.25, 81.75,
80.75, 80.50, 79.75, 78.25, 78.25, 78.25, 78.00, 77.75, 76.75, 76.25, 76.00,
75.25, 75.00, 74.50, 74.00, 73.50, 72.00, 71.25, 70.25, 69.50, 67.75, 67.25,
66.75, 66.75, 65.75, 65.50, 65.00, 64.50, 63.75, 63.75, 63.00, 61.00, 60.00,
59.75, 58.75, 57.75, 57.75, 57.50, 56.75, 56.25, 54.75, 54.00, 53.50, 53.25,
50.75, 50.50, 40.50, 40.25, 39.00, 36.25, 35.75

n = 73
mean = 71.9623
median = 74
```

The scores ran a little lower than I would have liked. After taking all factors into consideration **I've decided to add six points to all scores**. The score you'll see on D2L will be six points higher than the score written on the cover page of your exam.

CSC 372 Final Exam Solutions
Tuesday, March 12, 2015

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.*

   Here are some of my jots...

   Story of Richard Hamming's three questions for new hires at Bell Labs.

   "I think programming languages are the key to productivity."

   "Perl is the only popular language I hold in lower esteem than C."

   Recommended book *The Innovator's Dilemma*.

   Academics are batting zero for popular languages. [But I'd say Haskell is a still-rising counter-example.]

   Thinks Visual Basic was most influential language of the 1990s.

   Statistics gathered by Microsoft's Watson tool (not to be confused with IBM's Watson!) show that about 50% of crashes are caused by 1% of the bugs.

   His mention of "flight data recorders" for programs reminded me that Adobe had something like that in the works for its ActionScript/Flash Player stack. I've got a distinct memory of seeing a demo of it but a quick Google just now didn't turn up anything that looked familiar.

   He said that Go is the fastest growing language right now. It was also interesting to hear about a single code formatting standard being enforced by the Go formatter. There are benefits to freedom from choice, and I sure like a one-for-all-formatter better than Python's enforcement of indentation rules.

   And last but not least there was that incredible story about the birth of Erlang.

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?*

   I believe he'd give it a 5, but I counted a 4 or 5 as correct.

Problem 1: (6 points) <u>mean = 5.12, median = 6.00</u>

Cite three things about programming languages you learned by watching your classmates' video projects. Each of the three should be about a different language.

I tallied the languages cited in the responses and came up with the numbers below. I believe Mr. Nelson's video is destined to become a cult classic, so I gave it its own category, and didn't count it toward Ruby. I especially loved how he fit ten minutes of video into seven minutes by simply speeding it up, taking a lesson from Alvin and the Chipmunks.

Swift	27
Python	26
Ruby	17
Go	14
The Hardest Unsolved Problem in Computer Science	12
Java	10
Haskell	8
PHP	8
Lisp	6
C++	6
C	5
JavaScript	4
Erlang	4
Icon	3
Lisp	3
Perl	3
io	2
MATLAB	2
C#	2
Mathematica	2
F#	1

Lots of people commented about the ability to have emojis in Swift but just like Swift, Java allows programs to be written in Unicode. You can thus use emojis in Java, too; the question is whether your IDE or editor displays the full Unicode character set properly.

Problem 2: (3 points) mean = 1.09, median = 1.00

(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`*."*

Calling `f` with a `Char` produces a function that can called with a `Bool` to produce a final result, of type `Char`. `f` is most likely defined in curried form.

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

```
((((f  1) 2) g) + (x x)) + (((f g) f) 1)
```

Problem 3: (6 points) mean = 2.48, median = 2.00

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.*

```
longpos strings = snd $ head $ reverse $ sort $ zip (map length strings) [1..]
```

Problem 4: (9 points) mean = 6.82, median = 8.00

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.

```
sum = 0
while (print "Number? "; line = gets)
    line.chomp!

    if line =~ /^\d+$/
        sum += $&.to_f
    elsif line =~ /^((\d+) )?(\d+)\/(\d+)$/ then
        sum += $2.to_i + $3.to_f/$4.to_f
    else
        puts "Ignored: #{line}"
    end
end
puts "\nTotal: #{sum}"
```

Problem 5: (9 points) mean = 7.24, median = 8.50

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.*

```
items = {}

f = File.open("buy.data")

while line = f.gets
    type,item,f3,f4 = line.chomp.split("|")
    if type == "item"
        items[item] = [f3, f4.to_f]
    else
        items[item][1] *= 1 - f3.to_f*0.01
    end
end
f.close
```

```
total = 0
for item in ARGV
    if !items[item]
        puts "Price check for #{item}!\n"
        exit(1)
    else
        cost = items[item][1]
        printf("%s...%.2f\n", items[item][0], cost)
        total += cost
    end
end

printf("Total: $%.2f\n", total)
```

I was appalled by the number of students who used this structure, which turns an *O(n)* computation into an $O(n^2)$ computation:

```
for item in ARGV
    read buy.data, looking for item
```

Problem 6: (6 points) mean = 4.72, median = 6.00

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

```
module Outin # as a mixin...
    def outin
        a = self.dup
        while a.size > 0
            yield a[0]
            a = a[1..-1].reverse
        end
        nil
    end
end
```

Some students imagined that an iterator can be be called recursively but that doesn't work. However, I let those pass.

Problem 7: (5 points) mean = 2.87, median = 3.00

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.

```
class Fixnum
    def *(rhs)
        if rhs.class == String
            rhs * self
        else
            self * rhs
        end
    end
end
```

Problem 8: (10 points) mean = 8.16, median = 9.00

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.*

```
f(3).
f(4).
```

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

```
head2([H|_],[H,H]).
```

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

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

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

```
sum([],0).
sum([H|T],Sum) :- sum(T,Sum0), Sum is H + Sum0.
```

(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(L,Init) :- append(Init,[_],L).
```

Problem 9: (4 points) mean = 2.91, median = 3.00

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.
```

```
zip([H1|_],[H2|_], pair(H1,H2)).
zip([_|T1],[_|T2], R) :- zip(T1,T2,R).
```

Problem 10: (6 points) mean = 4.15, median = 5.00

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.
```

```
swapends([First|T],[Last|Rest]) :-
      append(Middle,[Last],T), append(Middle,[First],Rest).
```

I was very impressed with the elegance and symmetry of answers by Mr. Britton,

```
swapends([H|T],[R|F]) :- reverse(T,[R|RS]), reverse([H|RS],F).
```

and Mr. S. Stephens:

```
swapends([X|T],[Y|S]) :- reverse(T,[Y|R]), reverse(S,[X|R]).
```

In case you're wondering, yes, I had to type those in to convince myself they worked.

Mr. Wolfe had an elegant recursive solution: (minus an overlooked unification)

```
swaphelp([X],[Y],Y,X).
swaphelp([X|XS],[Y|YS],In,Out) :- X = Y, swaphelp(XS,YS,In,Out).
swapends([X|XS],[Y|YS]) :- swaphelp(XS,YS,X,Y).
```

Problem 11: (6 points) mean = 2.98, median = 4.00

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.
```

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

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

```
inout(L,R) :-
    findall(Val, outin(L,Val), Vals), reverse(Vals,RVals), member(R,RVals).
```

A common short but wrong answer was to simply reverse the list and call `outin` on the reversed list.

Problem 12: (12 points) **mean = 6.06, median = 7.00**

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. ...*

```
cross(Planks,Distance,Solution) :-
      layplanks(Planks, 0, Distance, Solution).

layplanks(_,Current,Distance,[]) :- Current >= Distance.

layplanks(Planks, Current, Distance, [Plank|MorePlanks]) :-
    select(Plank, Planks, LeftOver),
    NewEnd is Current + Plank,
    \+overpit(NewEnd),
    layplanks(LeftOver,NewEnd,Distance,MorePlanks).

overpit(N) :- pit(Start,Width), End is Start + Width, N >= Start, N =< End.
```

A common mistake was to have code that looked like this,

```
..., pit(Start,Width), End is Start, NewEnd < Start, NewEnd >= End, ...
```

where I've got `\+overpit(NewEnd)` above. With code like that you're asking, "Is there is **any** pit that the joint is not over?"

Problem 13: (7 points) (one point each unless otherwise indicated) **mean = 5.09, median = 5.00**

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?

Structures, but lots of people said "atoms".

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

When a predicate detects a condition that guarantees the predicate should fail. Or, "My final answer is no!"

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

`append(A,B,A)` can only be true if `B` is an empty list. Perhaps the programmer thinks that `append` works like `A = A + B`.

(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)*

The hallmark of an expression is that it produces a value but predicates can only succeed or fail. Any values produced by a predicate must be via instantiation.

In retrospect I decided this wasn't a very good question. Any non-trivial answer for it was counted as correct.

(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)*

`member` can both test for membership and generate values.

Problem 14: (5 points) (one point each unless otherwise indicated) **mean = 5.95, median = 6.00**

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.

Here's what I see:

The big one: Sather appears to be statically typed.

Types follow the entity whose type is being declared.

`|val1, val2, ..., valN|` appears to be array initializer syntax but there's also `#ARRAY{FLT}(|0.0|)`; the distinction between the two is not clear. `#ARRAY` is apparently the same kind of thing that `#OUT` is.

`reduce(bind(sqsum(_,_)))` implies that reduce is a higher-order function, but `bind` is apparently needed to create an intermediate value that refers to `sqsum`.

Last value in method does not implicitly become return value; explicit `return` statement is used instead.

One form of initialization is `:=`.

`#OUT` is a special value that produces output when a value is concatenated with it.

String literals are enclosed in double quotes and have backslash escapes.

Semicolons are required to terminate statements.

`class` may imply that Sather is object-oriented. I believe that only Mr. Bernth mentioned that `.append` implies object-orientation, too.

A number of students said that the free-standing `s` in `sqsum(s, e:FLT)` implied that types need not be specified, but in fact the type simply follows the identifier(s). (Note that in the Java declaration `int x, y;` the type `int` applies to both `x` and `y`.)

A handful of students said that `:` was cons, but I see no grounds for that whatsoever.

Problem 15: (6 points) mean = 5.46, median = 6.00

Answer the following general questions.

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

The type of every expression can be determined at compile time.

(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)

Symbolic forms allow more concise expression and that's good for common operations like concatenation and indexing.

Sometimes using a symbol avoids a naming problem. For example, Icon's unary `*` operator, which returns the length of a string or a list, the number of key/value pairs in a table, the numbers of characters in a character set, and more. Instead of deciding between `length`, `size`, `count`, `cardinality` (true story), etc., `*x` provides a helpful indefiniteness.

Designers must also consider whether there's a common underlying idea that makes a symbolic form appropriate, like using `+` for both addition and concatenation, although the latter is not commutative.

(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)

I tallied the answers. Here are the numbers:

Haskell:	16
Ruby:	24
Prolog:	28

The rationales for retention were myriad. Among them were which language was most interesting, which language would likely be most useful and/or marketable, which language had the most concepts useful in other settings, which language was least likely to be independently learned, and which language "I don't ever want to learn again."

Extra Credit Section (½ point each unless otherwise noted) mean = 2.85, median = 3.00

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

The first example in the *The Java Programming Language* by Ken Arnold and James Gosling had "`Sytem.out.println(...)`" in the first example.

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

The first round of CS host names were font names. Dr. Peter Downey found "Lectura", and it seemed perfect for an instructional machine. Some others were Bocklin, Megaron, Caslon, and Bembo.

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

Maybe the Prolog 500—the list appears to have about 502 entries.

(4) What does DWIM mean?

"Do what I mean." Runner-up, by P. Martin: "Drunken whales injure millions."

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

"Programming Languages Lab" is on the plaque beside the door, and Mr. Hickey knew that, but here are other titles:
The Lion's Den
The Den of Crushed Freshmen
Ephiphany H.Q.
Home [two entries!—whm]
The Thunderdome
A long staircase away.
the Lab
CS Dungeon
The "Please Help me" Lab
"When you're desperate"
The Help me room
Late-night Debugging Center
Big Poppa's House
The Hurt Locker
Last minute disaster
Panic Bunker
The Romper Room
Where the Problems Are Solved
This is where you go to finish your program.
Buckle your Seat Belts Room
Help Center for Sick 372 Students
The Meat Grinder
A room where people inside cry 9 times a semester.
Help Center
think tank
The Simpsons
whm's coder clubhouse
The Cool Kids Club
Broken Elevators
Windowless Dungeon of Office Hours [submitted by a fellow correctional officer]
Office hours central
my second home
Mitchell's Lab (?) [that does have a nice ring!]
The Last Minute Help Room
"The assignment is due tonight?!"
The computer lab that eats all of your time away.
The "Help Me" group
The penthouse
my apartment
Gould Simpson 942 (spelling counts) [ouch!]

Names for the building, apparently:
My girlfriend calls it the "Nerd Palace".
Palace of Science

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

My word is "strings", and Mr. Nelson came up with that, too.

J. Naranjo and T. Romero said "threading".

E. Harris, C. Jensen, C. Johnson, R. Macdonald, B. Streeter, and C. Troy all came up with "patterns", and that's surely true!

K. Enami pointed out that both SNOBOL4 and "weaving" have an "n" in them.

K. Hudspeth was creative with "weavematical". (Zero Google hits just now.)

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

1971

(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?

Ralph had a desk, but no chair! However, any answer was counted as correct. Twenty-two said "desk", twenty-six said "chair(s)", and one said "coffee maker".

Here's one last story about Ralph. As dial-up modems started becoming popular the phone company started worrying about modem users monopolizing circuits. There was a Tucson TV news story about this in the early 70s in which a phone company representative talked about the problems that heavy modem users could create. As a case in point they cited a customer in town whose modem had been continuously connected for over a month! Ralph told me that story and added, "That customer was me."

(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?

The first answer I read was by Mr. Dimka: "One person considers it their favorite language." Interestingly, that's an answer I had in mind, too. I believe I mentioned in class that I discovered the language RATSNO (Rational SNOBOL) among the Icon stuff. Dave Hanson developed it as an experiment in software adaptability, to see how hard it would be to retarget the RATFOR preprocessor (about 1500 lines), which had a design principle of "RATFOR does not know any FORTRAN". Hanson completed the conversion in about 8 hours.

I loved RATSNO and used it for several projects as an undergraduate at N.C. State, including a Pascal compiler. I've never gone looking for other users (I should!), but as I far as I know, I was the only regular user of RATSNO. But for me, RATSNO was a success. http://www.cs.arizona.edu/classes/cs372/spring15/csc412project.pdf is an example of some RATSNO code. It solves language equations using Arden's Lemma. You'll see that thirty-five years ago I wrote more comments than I do now.

Here are other answers that caught my eye:

"A successful language displaces a previous language."—S. Stephens
"It can solve a problem."—J. Weiser
"If a lot of people are using it without complaining."—N. Gelo
"People use it."—J. Tom
"Mother's approval."—M. Curry
"Helped solve one problem better than any other language."

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

I've pondered this question a fair amount over the years. During the Icon segment of 372 in Fall 1996, Larry Johnson said, "I'm just worried we're studying some dead language just because it was invented at the U of A.".

As a practical matter I think it takes a lot to kill a programming language. One of my first thoughts was that if a language is written in assembler and no machines with that architecture still exist, that language is surely dead. However, one could write an emulator for that architecture and revive the language. Even if all implementations were lost but specifications or programs remained, again the language could be revived.

I think I may have mentioned that there was never a implementation of the Alphard language but several papers were published about it. (Ralph thought that was "nuts!")

Here are some of the many answers I enjoyed:
"No one considers it their favorite."—A. Dimka
"No one can write a program in it without Googling."—J. Wolfe
"When there are no new programs being written in it."—C. Johnson
"When you can't Google it."—M. Justice
"When the tools to run the programming language no longer exist."—J. Wynne
"When it has no more followers."—E. Saetren
"They don't die; they just go to sleep for a very long time."—B. Hammond
"When it has C in the name."

Statistics and an adjustment

While grading I ultimately came to not like problems/questions 2(1), 13(1), and 15(2). Five points were added to all scores to compensate for those questions. The scores written on your exams do not reflect that five-point adjustment.

Here are all scores, with that five-point adjustment applied:

```
107.00, 106.50, 106.50, 106.00, 105.25, 102.50, 100.50, 97.00, 96.50, 96.00,
93.00, 91.00, 90.50, 90.00, 89.50, 89.50, 89.50, 88.00, 87.50, 87.00, 87.00,
86.50, 86.50, 86.00, 84.50, 83.50, 83.00, 83.00, 83.00, 82.50, 82.00, 81.00,
80.50, 79.50, 78.00, 78.00, 76.50, 76.50, 76.00, 75.50, 74.50, 74.00, 73.50,
73.50, 72.00, 72.00, 69.50, 68.00, 66.00, 64.50, 64.50, 63.00, 62.50, 62.00,
62.00, 61.50, 61.50, 61.50, 60.50, 60.50, 57.75, 56.50, 54.50, 51.00, 37.50

n = 65
mean = 78.9615
median = 80.5
```

All overall final averages for the course are shown below, with grade lines drawn in. You can find your final letter grade in 372 by looking at your "Overall average" on D2L and seeing where you fall in the list below.

Drawing grade lines is always tough. For those just short of a letter grade I double-checked his/her exam for any additional possible points.

```
------ A's below
104.70
104.26
104.24
103.50
102.92
101.96
101.53
101.23
100.15
100.00
96.83
96.75
```

```
96.67
96.39
96.38
96.04
95.87
95.51
94.54
93.85
93.57
92.92
92.87
92.65
91.44
90.58
89.99
89.88
------ A's above; B's below
88.89
87.97
87.22
86.15
86.02
85.16
84.15
82.41
82.25
82.11
82.05
80.33
79.57
----- B's above; C's below
77.74
77.70
77.69
76.94
74.58
72.35
70.85
70.59
69.98
69.87
69.20
69.09
68.49
----- C's above; D's below
66.95
65.54
64.69
64.32
63.61
60.06
----- D's above; E's below
53.28
52.83
31.09
30.67
```