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.)