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

READ THIS FIRST

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

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

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

If you are unsure about the form or operation of a language construct that is central to a problem's solution, you are strongly encouraged to ask the instructor about it.

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

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

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

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

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

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

Name: ______________________________________

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

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

What is the type of the following ML expression?

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

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

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

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

Cite a fundamental benefit of inheritance in C++.

What's unusual about this Prolog predicate?

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

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

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

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

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

What would be printed in response to the following query?

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

Write the `append/3` predicate.

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

Problem 2 (31 points):

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

The data to be processed looks like this

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(1) Write a routine `do_case(s)` that assumes `s` is a "`case...`" line and that returns a list that contains three values: A list of lists representing the cables, an integer representing the length to span, and a string that holds the desired endpoints.

(2) Write a routine `do_config(s)` that processes a cable configuration line and produces a list of lists representing the cables.

(3) Write a routine `check_sets(In,Out)` that takes two lists of cables and tests to be sure that the cables in `Out` are a subset of the cables in `In`.

(4) Write a routine `check_config` to check other aspects of the configuration.

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

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

[Space for solution for problem 2]

[More space for solution for problem 2]

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

Problem 3 (8 points):

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

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

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

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

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

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

yes
```

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

Problem 4 (6 points):

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

Example:

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

Problem 5 (6 points):

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

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

yes
```

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

More examples:

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

yes

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

no

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

yes

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

yes

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

no
```

Problem 6 (6 points):

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

Problem 7 (4 points):

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

Problem 8 (4 points):

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

Problem 9 (3 points):

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

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

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

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

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

- f3 20;
val it = 10 : int

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

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

Problem 10 (4 points):

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

Example:

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

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

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

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

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

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

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

Problem 11 (8 points):

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    PrintableMList m2(5);

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

    m2.print();

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

Output:

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

[Space for solution for problem 11]

EXTRA CREDIT SECTION

EC 1 (1 point):

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

EC 2 (1 point):

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

EC 3 (1 point):

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

EC 4 (1 point):

Name a language that is an ancestor of Icon.

EC 5 (1 point):

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

EC 6 (1 point):

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

EC 7 (1 point):

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

EC 8 (1 point):

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

EC 9 (1 point):

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

EC 10 (1 point):

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