Name:____________________________________ Seat row **and** number: __________

CSc 372, Fall 2001
Final Examination
December 14, 2001

READ THIS FIRST

Fill in your name and seat row/number above.

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

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

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

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

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

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

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

You need not include any explanation in an answer if you are confident it is correct. However, if an answer is incorrect, any accompanying explanation may help you earn partial credit.

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

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

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

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

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

Problem 1: (7 points)

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

These buffer contents:

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

would be changed to this:

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

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

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

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

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

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

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

```
    (goto-char 1)
         1         2         3         4         5         6
12345678901234567890123456789012345678901234567890123456789012345
    (while (not (= (point) (point-max)))
        (setq char (char-after (point)))
```

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

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

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

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

A blank page follows.

(Space for solution for problem 2)

For reference:

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

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

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

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

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

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

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

then this:

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

then this:

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

then this:

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

and so forth.

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

A blank page follows.

(Space for solution for problem 4)

Problem 5: (9 points)

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

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

Problem 6: (6 points)

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

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

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

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

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

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

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

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

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

Problem 7: (2 points)

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

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

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

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

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

Problem 8: (8 points)

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

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

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

Problem 9: (8 points)

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

Examples:

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

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

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

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

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

Problem 10: (10 points)

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

Problem 11: (10 points)

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

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

Problem 12: (10 points)

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

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

(B) Icon has several polymorphic operators and functions. For example, the unary `*` operator returns the number of elements in an object. The `delete` function removes elements from sets and tables. What are some tradeoffs that a language designer must take into account when considering inclusion of a polymorphic operation in a language?

(C) Imagine a language that has heterogeneous lists like Prolog, Icon, and Lisp, but that also has a tuple type that is very similar to ML. Present an argument either for or against the claim that if a language has heterogeneous lists, tuples provide no additional benefit.

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

(E) The instructor said that "every programmer should have a language like Icon in their toolbox". What are the characteristics of Icon, and similar languages, that make it worthwhile to know such a language in addition to mainstream languages like Java and C++? (Or, argue that knowing a single language like Java or C++, and knowing it well, is all that's needed.)

(F) A language implemented in C, such as Emacs Lisp, typically has some library functions coded in C and the balance coded in the language itself. What factors influence whether a function is implemented in the language itself?

A blank page follows.

(Space for answers for problem 12)

EXTRA CREDIT SECTION (one point each)

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

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

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

(d) As of midnight on December 12, how many messages have been sent to the CSc 372 mailing list? (Your answer must be within 10%.)

(e) The instructor has immediate plans to rewrite the Icon assignments in another language to see how that language compares to Icon. What's the language?

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