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

READ THIS FIRST

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

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

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

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

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

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

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

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

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

Name: _____

Problem 1 (12 points):

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

Problem 2 (8 points):

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

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

Problem 3 (5 points):

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

Examples:

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

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

Problem 4 (3 points):

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

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

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

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

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

Problem 5 (7 points):

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

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

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

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

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

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

```
?- integer(5).
yes

?- integer(a).
```

```
no

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

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

Problem 6 (6 points):

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

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

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

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

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

Problem 7 (10 points):

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Problem 8 (8 points):

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

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

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

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

Examples (with `ie`):

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

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

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

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

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

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

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

Problem 9 (6 points)

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

Examples of words satisfying the desired condition:

```
acrobatic

elasmobranch

swashbuckler
```

Problem 10 (6 points)

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

Examples (with `ie`):

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

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

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

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

Problem 11 (7 points):

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

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

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

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

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

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

Problem 12 (4 points):

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

Example:

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

Problem 13 (9 points):

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

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

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

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

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

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

Problem 14 (9 points)

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

A private default constructor.

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

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

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

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

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

EXTRA CREDIT SECTION

EC 1 (5 points):

Name five programming languages that originated before 1985.

EC 2 (5 points max):

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

EC 3 (1 point):

What is the instructor's favorite programming language?

EC 4 (1 point):

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

EC 5 (2 points)

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

EC 6 (3 points)

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

EC 7 (1 point)

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

EC 8 (2 points)

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

EC 9 (5 points):

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

EC 10 (2 points):

Why are static class members an essential element of Java?

EC 11 (1 point):

What is the Prolog 1000?