CSc 372, Spring 1996
Mid-Term Examination
Tuesday, March 5, 1996

READ THIS FIRST

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

This examination consists of 20 problems presented on 12 numbered pages. When you are told to begin, first check to be sure you have all the 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.

Please feel free to ask the instructor questions during the course of the exam. If you are unsure about the form or operation of a language construct that is central to a problem's solution, you are especially encouraged to ask the instructor about that construct.

If you're completely puzzled on a problem, the instructor may offer you a hint, at the cost of a deduction of points. If you agree to accept such a hint, the instructor will note that deduction on your paper.

Try to avoid leaving a problem completely blank—that will certainly earn no credit. If you can offer any sort of pertinent response it may earn you partial credit.

Problems marked with an asterisk (*) disproportionately hard with respect to their point values. You should do such problems last.

Except as otherwise noted, you may use any language elements you desire. For the Icon problems you may use `split`.

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

Problem 1 (6 points):

State a definition for the term "programming language".

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

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

Problem 2 (4 points):

Write ML expressions having the following types:

```
int * string list
```

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

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
```

```
g: int -> int -> int list
```

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`?

`g`?

`h`?

`a`?

`b`?

`x`?

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.

Problem 5 (3 points):

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

What is one possible implication of the warning?

Write an ML function that would produce that warning.

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?

y?

z?

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!)

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

Problem 7 (5 points):

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

```
- length;
val it = fn : int list -> int

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

You may not use the built-in `length` function!

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`.

You may wish to use the function `real`, of type `int -> real`, to convert an `int` into a `real`.

Examples of usage:

```
- avglen;
val it = fn : 'a list list -> real
- avglen([]);
uncaught exception EmptyList
- avglen([[]]);
val it = 0.0 : real
- avglen([[],[1]]);
val it = 0.5 : real
- avglen([[1],[1],[2,2]]);
val it = 1.33333333333333 : real
- avglen([[true,true],[true],[true,true,true]]);
val it = 2.0 : real
```

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`.

Examples of usage:

```
- F_L_eq;
val it = fn : ''a list -> bool
- F_L_eq([]);
val it = false : bool
- F_L_eq([1]);
val it = true : bool
- F_L_eq([1,2,1]);
val it = true : bool
- F_L_eq([1,2]);
val it = false : bool
- F_L_eq([[],[1],[2],[],[]]);
val it = true : bool
- F_L_eq(explode "abcde");
val it = false : bool
```

You may NOT use a function that reverses a list.

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`.

For example,

```
f([f1, f2, f3, ..., fM],[x1, x2, ..., xN])
```

would produce a value equivalent to an expression like this:

```
[[f1(x1), f2(x1), ..., fM(x1)],
 [f1(x2), f2(x2), ..., fM(x2)],
 ...
 [f1(xN), f2(xN), ..., fn(xN)]]
```

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?

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

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
```

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

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`.

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.

`Point` is not required to handle negative coordinates, but if it can, that is fine. That is, for the call `Point("(-10,-20)")`, it is acceptable either if `Point` fails or if it returns the list `[-10,20]`.

Examples:

```
][ Point("(1,2)");
   r := L1:[1,2]  (list)
][ Point("(100,200)");
   r := L1:[100,200]  (list)
][ Point("10,20");
Failure
][
```

Problem 15 (8 points)

Write an Icon program that reads standard input and produces a histogram of line lengths encountered. For example, for the file

```
abc
def
ghij
klm
nop
qr
st
uv
wxyz
```

The following output would be produced:

```
Length  Occurrences
2       ***
3       ****
4       **
```

If you choose to use a table in your solution, note that if `x` is a table, `sort(x)` produces a list of the form `[[k1,v1],[k2,v2],...,[kN,vN]]` where each `ki/vi` represents a key/value pair in the table. The pairs are ordered by increasing value of `ki`.

Problem 16 (9 points):

Imagine a file with a format such as this:

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

Each line begins with a sequence number that is followed by a dot. An arbitrary string follows the dot. Note that there is no whitespace at the beginning of the line.

In a given file all sequence numbers are the same length, but the sequence number length may vary from file to file—do not assume a length of two for sequence numbers. Sequence numbers are always specified with leading zeros.

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.

The length of the output lines is determined by a command line argument, which if not specified defaults to 10.

Assuming that the program is called `assemble`, here are some invocations on an input file containing the above three lines:

```
% assemble <assemble.in
just testi
ng this ou
t
% assemble 15 <assemble.in
just testing th
is out
% assemble 1000 <assemble.in
just testing this out
%
```

Another input file and sample invocations:

```
% cat assemble.in2
0004.xy
0001.abcdefg
0002.hijklmnop
0005.z...
0003.qrstuvw
% assemble 25 < assemble.in2
abcdefghijklmnopqrstuvwxy
z...
```

Problem 16 (space for solution):

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.

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.

Examples:

```
][ size("");
   r := 0  (integer)
][ size("abc");
   r := 3  (integer)
][ size([1,2,3,4]);
   r := 4  (integer)
][ size(table(""));
   r := 0  (integer)
```

Problem 19 (6 points) (*):

Write an Icon program to read standard input and print out the largest integer found in the input. Consider an integer to be simply a series of consecutive digits; don't worry about issues with negative numbers. The program's output should be simply the largest integer. If no integers are found the program should output "`No integers`".

```
% bigint
On February 14, 1912, Arizona
became the 48th state.
^D
1912
% bigint
A test with no integers
in the input.
^D
No integers
% bigint
-100-1000-100000
^D
100000
```

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.