CSc 372, Spring 1996
Mid-term Examination Solution Key

Problem 1 (6 points):

State a definition for the term "programming language".

In the slides, a programming language is said to be "a notation for the description of computation".

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

Ideal answers included "a looping construct" and "the ability to change values via assignment".

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

Ideal answers included "recursion", "functions", and "use of functions as values".

Problem 2 (4 points):

Write ML expressions having the following types:

```
int * string list

     (1, ["x"])

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

     (1, (1, [(1,1)]))
```

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

     fun g(1) = 1
     fun f(g, 1) = g(1) = 1

g: int -> int -> int list

     fun f 1 2 = [1, 2]
```

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?   string * int list * (string -> int) -> bool

g?   string -> int

h?   int list -> bool

a?   string

b?   int list

x?   int
```

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.

```
exception NotOne;
fun f(1) = 1
 |  f(n) = raise NotOne
```

Problem 5 (3 points):

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

There is a type-compatible tuple that is not matched by any of the patterns defining a function.

What is one possible implication of the warning?

A match exception may be encountered when the function is executed.

Write an ML function that would produce that warning.

```
fun f(1) = 1
```

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?   [[1]]

y?   (2, [3])

z?   [4, 5, 6]

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!)

```
fun (x,y,z::xs) = 21
```

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

```
'a * 'b * 'c list -> int
```

Problem 7 (5 points):

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

```
fun length([]) = 0
 |  length((_:int)::xs) = 1 + length(xs);
```

Many persons defined a length function with type `'a list -> int.` Others, in trying to force the `int list` type ended up with a function that summed the elements in the list instead of counting them.

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`.

```
fun avgLen(L) =
     let
          fun sum([]) = 0
           |  sum(x::xs) = x + sum(xs)
          val lens = map length L
     in
          if length(L) = 0 then raise EmptyList
            else real(sum(lens)) / real(length(L))
```

This problem was not intended to be difficult but in fact it had a relatively low success rate. The key observation is that the essence of the problem is calculate the average of a list of integers.

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`.

```
fun F_L_eq([]) = false
 |  F_L_eq([x]) = true
 |  F_L_eq(x::xs) =
         let
             fun last([x]) = x
              |  last(_::xs) = last(xs)
         in
             x = last(xs)
         end
```

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`.

```
fun f(FL, []) = []
 |  f(FL, v::vs) =
     let
          fun apply_each([], x) = []
           |  apply_each(f::fs, x) = f(x)::apply_each(fs, x)
     in
          apply_each(FL, v)::f(FL, vs)
```

```
        end
```

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?

Lists in ML must be homogeneous—all elements must be of the same type. Elements in an Icon list can be of differing types.

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

```
[1, 1.0]
```

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

     fun f(g::gs) = g("x") + 1;

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

     fun g "" 1 2.0 = [true]
```

Another way:

```
     fun g s i r = [s = "x", i = 1, r = 1.0];
```

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`.

```
procedure Reverse(s)
        r := ""
        every r := !s || r
        return r
end
```

Another way:

```
procedure Reverse(s)
        r := ""
```

```
            every i := 1 to *s do
                    r := s[i] || r
            return r
    end
```

I had intended Reverse to be a trivial problem to serve as a warm-up for the Icon portion of the exam, but only a handful of persons produced a solution that would actually work.

A number of persons produced solutions that used list manipulation functions such as push and pull to manipulate strings, but in fact those functions don't work on strings. Despite that, such solutions typically received full credit.

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.

```
procedure Point(s)
    s ? {
            ="(" &
            x := tab(many(&digits)) &
            ="," &
            y := tab(many(&digits)) &
            =")" & pos(0) &
            return [integer(x),integer(y)]
        }

end
```

A number of persons tried to solve this with split, but if split was used you needed to use the three-argument form and examine each piece produced. Most of the split-based solutions would accept strings such as ",,2,,,3,," or "()()2()3()", or worse.

Problem 15 (8 points)

Write an Icon program that reads standard input and produces a histogram of line lengths encountered.

```
procedure main()
    hist := table("")

    while line := read() do
        hist[*line] ||:= "*"

    every pair := !sort(hist) do
        write(pair[1], "\t", pair[2])

end
```

Problem 16 (9 points):

Imagine a file with a format such as this:

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

...

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.

```
procedure main(args)
    len := args[1] | 10

    lines := []

    while line := read() do
        put(lines, line)

    lines := sort(lines)
    out := ""
    every line := !lines do {
        out ||:= (line ? ( tab(upto('.')+1) & tab(0) ))
        }

    out ? {
        while write(move(len))
        write(tab(0))
        }
end
```

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.

```
procedure cons(x,y)
     return [x] ||| y
end
```

It was also satisfactory to use a solution such as this:

```
procedure cons(x,y)
     return push(y,x)
end
```

if it was noted that the approximation was poor because list `y` is changed as a side-effect.

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.

```
procedure size(x)
    count := 0
    every !x & count +:= 1
    return count
end
```

Problem 19 (6 points) (*):

Write an Icon program to read standard input and print out the largest integer found in the input. ...

```
procedure main()
    while line := read() do {
        line ? while tab(upto(&digits)) do {
                val := tab(many(&digits))
                if /maxval | (val > maxval) then
                    maxval := val
                }

        }

    write(\maxval|"No integers")
end
```

Several persons approached this problem with `split`, but it is necessary to split on `~&digits` rather than whitespace to achieve a correct solution.

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.

An argument can be made either way and therefore, answering either "yes" or "no" earned a point. A reasonable argument of any sort earned two more points.

I would argue that most of the functional solutions we studied could be implemented in Icon with a minimum of difficulty, aside from Icon lacking an equivalent to ML's pattern matching facility. (It is said that Icon supports pattern matching via string scanning, but that is a completely different facility that shares the same name.)

On the other hand, Icon has nothing like anonymous functions, composition, or currying and therefore Icon's handling of functions as values, a cornerstone of functional programming, falls short.