Spring, 1995 Name: ____________________

Com S 541 — Programming Languages 1

Test on Functional Programming and SML

Special Directions for this Test

This test has 4 questions and pages numbered 1 through 4.
Before you begin, please take a moment to look over the entire test so that you can budget your time.
You should program in a purely functional style; for example, don't use `ref`. Other than that, you can use anything that is built-in to SML or listed on the inside of the back cover of Paulson's book. **You may use any other function in Paulson's book, or from your notes, but to do so you must write it into your answer.**

General Directions from the "Course Specification"

Short answer questions will be graded on the basis of completeness (did you list all the relevant parts of the answer?) soundness (did you make any errors of fact or contradict yourself?) and clarity. You can help ensure clarity by giving examples where possible.
For essays, critical justifications, and the like there may be no one right answer; so the criteria for judgement will be whether your essay is:

- clear (you use examples where appropriate, you write clear sentences, diagrams, and you avoid excess verbosity),
- sound (you start from facts or reasonable assumptions, your logic is convincing), and
- complete (you consider relevant aspects of the problem and alternatives).

The highest grade will be earned by an answer that is, in addition, concise and elegant; that is, answers that say the most with the least amount of excess. Extra consideration will be given to answers that are especially creative. For programming problems, there is usually more than one right answer, hence it is important to present with your program evidence that it is correct. Answers to programming questions will be scored on the following points:

- elegance and clarity (conciseness, using relevant techniques to modularize the solution, making good use of language features, the creativity of your solution, and how clear and convincing your argument is that the code is correct)
- correctness (whether your program solves the given problem correctly in the given programming language)

You should try to eliminate all syntax errors from your programs, as they will affect both the correctness and clarity of your solution; you may receive no points for programs with major syntax errors. Run-time space and time efficiency of programs is not important, unless the problem says so. Also, checking of external inputs, prompting, labeling output, and other conventions for dealing with humans are not important (for this class), unless the problem says so.
The breakdown of points between these categories is roughly as follows.

- 50% for elegance and clarity (using relevant techniques to modularize the solution, making good use of language features, the creativity of your solution),
- 50% for correctness (whether your program solves the given problem correctly in the given subset of the given language)

You are encouraged to use auxiliary subroutines whenever they are helpful.

1. (10 points) Consider the following functional.

```
fun itlist [] f b = b
  | itlist (a::lis) f b = f (a, itlist lis f b);
```

Complete the definition of the function `len: 'a list -> int`, which returns the length of its argument list, by filling in the blank below. (As examples, `(len [4.5, 7.2, 8.1]` = 3, and `len []` = 0.)

```
fun len lis = itlist _____________________________________________
```

2. (15 points) Recall the following definitions in SML.

```
datatype 'a seq = Nil | Cons of 'a * (unit -> 'a seq);
fun head (Cons(x,xf)) = x;
fun force f = f();
fun tail (Cons(x,xf)) = force xf;
fun consq (x,xq) = Cons(x, fn() => xq);
```

In this problem you are to write a curried function

```
  merge: ('a*'a -> bool) -> 'a seq -> 'a seq -> 'a seq
```

such that, assuming the elements of two sequences `s1` and `s2` are totally ordered by `c`, `(merge c s1 s2)` is a sequence that contains just the elements of `s1` and `s2`, also ordered by `c`. For example, with `takeq` and `from` as in Paulson's book:

```
takeq (5, merge (op <) (consq(3, consq(4, (consq(10, Nil)))))
                       (consq(1, consq(5, Nil))))
   = [1, 3, 4, 5, 10];
takeq (5, merge (op >) (consq(9, consq(6, consq(2, Nil))))
                       (consq(6, consq(1, Nil))))
   = [9, 6, 6, 2, 1];
takeq (4, merge (op >=) (consq(9, consq(9, Nil))) (consq(9, consq(1, Nil))))
   = [9, 9, 9, 1];
takeq (7, merge (op <) (from 1) (from 3)) = [1,2,3,3,4,4,5]
```

Your function should not perform any unnecessary computation.

3. (15 points) In this problem we will investigate representing environments by functions, with the type `Environ` below.

```
type Identifier = string;
type Bindable = int;
datatype EnvironCoDom = bound of Bindable | unbound;
type Environ = Identifier -> EnvironCoDom;
```

You are to define each of the following functions.

```
empty_environ : Environ
bind : (Identifier * Bindable) -> Environ
find : (unit -> Bindable) -> (Environ * Identifier) -> Bindable
```

Informally, `empty_environ` defines an empty environment, `bind(I,b)` creates a mini-environment in which `I` is bound to `b` and every other identifier is unbound, and `find f (env, I)` looks up the value (an `int`) bound to `I` in `env`, calling `f` if there is no binding. For example, the following equations should hold.

```
find (fn () => 0) (empty_environ, "x") = 0;
find (fn () => 0) (bind("anId", 3), "anId") = 3;
find (fn () => 0) (bind("anId", 3), "x") = 0;
```

4. Consider the following feature of a language named S. The language has a type constructor called **union**. One can use it to write a type of the form

$$\textbf{union of } T_1 + T_2,$$

where T_1 and T_2 are types. In this language one can give names to types, for example

type `IB` = **union of** `int` + `bool`

says that `IB` is a type whose values are either values of type `int` or `bool`. Thus 7 has type `int` and also `IB`. Similarly, `true` has type `bool` and also `IB`.

There is a built-in curried function in S, **isOfType** that takes a type T and returns a function that takes a value of a union type that includes T and returns true if the value comes from the T part of the union. For example, if `x` is an identifier declared to have type `IB`, then one can write

isOfType `int x`

This will be true, for example, when `x` denotes 7, but false if `x` denotes `true`. In the same situation, one can also write

isOfType `bool x`

to test whether `x` is from the a boolean part of the type `IB`.

(a, 5 points) What feature of SML is most closely related to **union** in S?

(b, 15 points) Criticize the **union** in S by describing three (3) things that can be done using SML's feature that can't easily be done with a **union** in S. Give examples to illustrate each. (Note: if you have a question about the semantics of S that is not described above, make an assumption about it and write down your assumption.)