Name: ______________________________

COP 4020 — Programming Languages I (Spring 2011)

Test on the Declarative Model

Directions for this test

This test has 9 questions and pages numbered 1 through 9.
This test is open book and notes, but no electronics.
Before you begin, please take a moment to look over the entire test so that you can budget your time.
Clarity is important; if your programs are sloppy and hard to read, you may lose some points. Correct syntax also makes a difference for programming questions.
When you write Oz code on this test, you may use anything we have seen in chapters 1–2 of our textbook. But unless specifically directed, you should not use imperative features (such as cells) or the library functions `IsDet` and `IsFree`.
Problems relating to the kernel syntax can only use features of the declarative kernel language.

For Grading

Question:	1	2	3	4	5	6	7	8	9	Total
Points:	10	10	10	10	10	15	15	10	10	100
Score:										

The first three problems ask for sets of free or bound variable identifiers. For these problems, write the entire requested set in brackets. For example, write $\{V, W\}$, or if the requested set is empty, write $\{\}$.
Also, recall that **declare** is not in the declarative kernel, so you should *not* imagine an implicit **declare** in the code given for these problems.
Note: Desugaring the statement or expression into kernel language may be helpful.

1. [Concepts] Consider the following Oz procedure:

```
proc {MyMap Xs F ?Ys}
   case Xs of
      nil then Ys = nil
   else
      case Xs of
         X|Xr then
         local Y Yr in
            Ys = Y|Yr
            {F X Y}
            {MyMap Xr F Yr}
         end
      end
   end
end
```

   (a) (5 points) Write down the set of free identifiers that occur in this piece of code.

   (b) (5 points) Write down the set of bound identifiers that occur in this piece of code.

2. [Concepts][MapToLanguages] Consider the following Java function:

```
public static int factorial(int num) {
    if (num == 0)
        return 1;
    else
        return num*factorial(num-1);

}
```

   (a) (5 points) Write down the set of free identifiers that occur in this piece of code.

   (b) (5 points) Write down the set of bound identifiers that occur in this piece of code.

3. [Concepts] Consider the following Oz procedure:

```
local S in
   P = proc {$ F R U}
          R = {F A}
       end
   local V in
      {P fun {$ A} A end V U}
   end
end
```

(a) (5 points) Write down the set of free identifiers that occur in this piece of code.

(b) (5 points) Write down the set of bound identifiers that occur in this piece of code.

4. (10 points) [Concepts] Which of the following are correct statements about tail recursion. (Circle the letters of all the following that are correct. Don't circle incorrect statements. There may be zero, one, two, or more correct statements.)

   A. A procedure is tail recursive only if whenever it makes a recursive call, there are no pending computations that need to be executed after such a recursive call returns.

   B. Procedures in Oz that are tail recursive can be executed without the stack growing due to their recursive calls.

   C. A procedure is tail recursive only if works on lists and only if it always makes a recursive call passing itself the tail of the list.

   D. No tail recursive function can be written in Oz.

   E. Procedures in Oz that are tail recursive can always be executed in a constant amount of time.

5. [Concepts]

Recall that in Oz, the partial value representing a procedure is called a *closure*. A closure is a pair consisting of the procedure's text along with the contextual environment. (An environment is a mapping from variable identifiers in the source code to entities in the store.)

(a) (5 points) Why is the contextual environment required to be part of the procedure value?

(b) (5 points) Give an example of an Oz procedure for which simply the text of the procedure would suffice i.e. no accompanying environment would be necessary to completely specify this procedure's value.

6. (15 points) [Concepts]

   Desugar the following into the declarative kernel language by expanding all syntactic sugars and linguistic abstractions. (If you run out of space, use the blank page at the back of this page.)

```
fun {IsEven X}
   if X == 0 then
      true
   else
      {IsOdd X-1}
   end
end
```

   Assume that the Mozart/OZ library built-in procedure `Value.'=='` takes three arguments, compares if the first two are equal, and binds the (boolean) result to the third argument. Similarly, the procedure `Number.'-'` takes three arguments, subtracts the second from the first, and binds the result to the third argument. Also, imagine that `IsOdd` is defined elsewhere as a procedure which takes two arguments, finds if the first argument is an odd number and binds the (boolean) result of this test to the second argument.

This page intentionally left blank

7. [Concepts] Consider the following Oz code:

```
declare Area
local R in
   R = 10.0
   local F in
      Area = proc {$ Res}
                local Pi in
                   Pi = 3.14
                   local Temp in
                      {Number.'*' R Pi Temp}
                      {Number.'*' R Temp Res}
                   end
                end
             end
      local R in
         R = false
         local Res in
            {Area Res}
            {Browse Res}
         end
      end
   end
end
```

(a) (5 points) When the above code runs in Oz, what will happen?

(b) (10 points) Suppose Oz used dynamic scoping. If this code were interpreted using dynamic scoping, what would happen when the code is run? (Give a brief explanation.)

8. [Concepts]

(a) (5 points) What is the output, if any, of the following code in Oz? Briefly explain why that output appears.

```
local SimpleRec = arecord(number: 4020 name: anon) in
   case SimpleRec of
      arecord(number: N name: S) then {Browse first#N#S}
   [] arecord(number: M name: T) then {Browse second#M#T}
   else {Browse third}
   end
end
```

(b) (5 points) What is the output, if any, of the following code in Oz? Briefly explain why that output appears.

```
local SimpleRec = arecord(number: 4020 name: anon) in
   local F2 = 2 in
      local T5 = 5 in
         case SimpleRec of
            arecord(number: F2 name: T5) then {Browse first#F2#T5}
         [] arecord(number: M name: T) then {Browse second#M#T}
         else {Browse third}
         end
      end
   end
end
```

9. (10 points) [Concepts]

Consider the following two Oz functions, `Length` and `IterLength`.

```
fun {Length Xs}
   case Xs of nil then 0
   [] _|Xr then 1+{Length Xr}
   end
end

fun {IterLength I Ys}
   case Ys
   of nil then I
   [] _|Yr then {IterLength I+1 Yr}
   end
end
```

`Length` can be used to find the length of a list `L` when invoked using `{Length L}`. `IterLength` can be used to find the length of a list `L` when invoked using `{IterLength 0 L}`. Both of them give the correct answer.

Briefly describe which function is more efficient in terms of the space requirements by comparing the sizes of their semantic stacks during execution.

.