Spring, 2010 Name: _____

COP 4020 — Programming Languages I

Test on the Declarative Model

Special Directions for this Test

This test has 9 questions and pages numbered 1 through 7.
This test is open book and notes.
If you need more space, use the back of a page. Note when you do that on the front.
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.
You are encouraged to define functions or procedures not specifically asked for if they are useful to your programming; however, if they are not in the Oz base environment, then you must write them into your test.

For Grading

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

1. (10 points) [Concepts] What happens when the following code executes in Oz? Briefly explain why that happens.

```
local N in
   N = 0
   local Sum in
      Sum = proc {$ I}
               if I > 0
               then N = N+1
               else skip
               end
            end
      {Sum 10}
      {Browse 'N is '#N}
   end
end
```

2. (5 points) [Concepts] [MapToLanguages] The programming language Ruby checks for type errors when the program is running, in general. So what kind of type checking does Ruby have? (Give the technical term for it.)

3. [Concepts]

   (a) (5 points) Are declarative programs in Oz (i.e., ones that could desugar to the declarative kernel language) necessarily deterministic? (Answer "yes" or "no" and give a brief explanation.)

   (b) (5 points) What is one advantage of using a deterministic language when writing concurrent programs (e.g., with multitheading)?

The next 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 $\{\}$.

4. Consider the following Oz statement in the declarative kernel language.

```
local A in
   A = 40
   local R in
      {MyFun A R}
      X = R
   end
end
```

   (a) (3 points) [Concepts] Write the entire set of the variable identifiers that occur free in the statement above.

   (b) (2 points) [Concepts] Write the entire set of the variable identifiers that occur bound in the statement above.

5. Consider the following statement in the declarative kernel language.

```
local MyMap in
   MyMap = proc {$ Ls F R}
              case Ls of
                 '|'(1:H 2:T) then local Res in
                                      {F H Res}
                                      local RestAns in
                                         R='|'(1:Res 2:RestAns)
                                         {MyMap T F RestAns}
                                      end
                                   end
              else R=nil
              end
           end
   {MyMap MyList MyFun MyRes}
end
```

   (a) (5 points) [Concepts] Write the entire set of the variable identifiers that occur free in the statement above.

   (b) (10 points) [Concepts] Write the entire set of the variable identifiers that occur bound in the statement above.

6. [Concepts] [MapToLanguages] Consider the following Java statement.

```
for (int j = 0; j < size(arr); j++) {
    res[j] = g(arr[j]);
}
```

(a) (5 points) Write below, in set brackets, the entire set of variable identifiers that occur free in the Java code above.

(b) (5 points) Write below, in set brackets, the entire set of variable identifiers that occur bound in the Java code above.

7. [Concepts] Consider the following Oz code in the declarative kernel language.

```
local A in
   local B in
      A = 40
      B = 20
      local F in
         F = proc {$ R} local Hundred in
                           Hundred = 100
                           local A100 in
                              {Number.'*' A Hundred A100}
                              {Number.'+' A100 B R}
                           end
                        end
          end
         local A in
            local B in
               A = true
               B = false
               local Res in
                  {F Res}
                  {Browse Res}
               end
            end
         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. (15 points) [Concepts] Desugar the following Oz code into kernel syntax by expanding all syntactic sugars, so that your answer is an equivalent statement in the declarative kernel language. (Assume that the identifier `Res`, and the function identifiers `Reverse` and `RevIter` are declared elsewhere. Do *not* implement `RevIter`.))

```
fun {Reverse Ls} {RevIter Ls nil} end
Res = {Reverse [4]}
```

9. [Concepts]

(a) (7 points) What is the output, if any, of the following code in Oz? Briefly explain why that output appears.

```
local MyEvent = downhill(number: 4020 slot: 15) in
   case MyEvent of
      downhill(number: N slot: S) then {Browse first#N#S}
   [] downhill(number: M slot: T) then {Browse second#M#T}
   else {Browse third}
   end
end
```

(b) (8 points) What is the output, if any, of the following code in Oz? Briefly explain why that output appears.

```
local MyEvent = downhill(number: 4020 slot: 15) in
   local F2 = 2 in
      local T5 = 5 in
         case MyEvent of
            downhill(number: F2 slot: T5) then {Browse first#F2#T5}
         [] downhill(number: M slot: T) then {Browse second#M#T}
         else {Browse third}
         end
      end
   end
end
```