Spring, 2009 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 6.
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 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:	8	15	5	6	16	15	10	10	15	100
Score:										

The first three problems ask for sets of free or bound variable identifiers that occur bound in the statement above. Write the entire requested set in brackets. For example, write $\{V, W\}$, or if the requested set is empty, write $\{\}$.

1. Consider the following Oz statement in the kernel language.

```
local X in
   local Y in
      X = 3
      {MyProc X Y}
      Z = Y
   end
end
```

   (a) (4 points) [Concepts] Write the entire set of the variable identifiers that occur free in the statement above.

   (b) (4 points) [Concepts] Write the entire set of the variable identifiers that occur bound in the statement above.

2. Consider the following Oz statement.

```
Mult = proc {$ N1 N2 Res}
       case N1 of
          succ(num: Pre) then
          local Val in
             {Mult Pre N2 Val}
             {Plus N2 Val Res}
          end
       else local Assg in
               Assg = proc {$ A B R} R = A end
               {Assg zero Pre Res}
            end
       end
    end
```

   (a) (6 points) [Concepts] Write the entire set of the variable identifiers that occur free in the statement above.

   (b) (9 points) [Concepts] Write the entire set of the variable identifiers that occur bound in the statement above.

3. [Concepts]

   (a) (3 points) Name a programming language that uses static type checking.

   (b) (2 points) Name a programming language that uses dynamic type checking.

4. [Concepts] Consider the following Java method declaration.

```
public void sum(int[] a, int n, int m) {
    for (int i = 0; i < n; i++) {
        total = total + compute(a[i]);
    }
}
```

(a) (3 points) Write below, in set brackets, the entire set of variable identifiers that occur free in the Java code above.

(b) (3 points) Write below, in set brackets, the entire set of variable identifiers that occur bound in the Java code above.

5. [Concepts] Consider the following Oz code.

```
local F in
   local ToInt in
      ToInt = proc {$ N ?R}
                 case N of
                    %% Parts (c) and (e) ask about the call below
                    succ(num: Pre) then R = 1 + {ToInt Pre}
                 else R = 0
                 end
              end
      F = ToInt
   end
   local Temp in
      %% Parts (b), (d), and (e) ask about the call below
      {F succ(num: succ(num: succ(num: succ(num: zero)))) Temp}
      {Browse Temp}
   end
end
```

(a) (2 points) When the above code runs, what output, if any, appears in the browser?

(b) (4 points) At the point of the call of F on line 14 (just below the second comment), is there a binding for `ToInt` in the current environment? Give a brief explanation.

(c) (4 points) Will the call to `ToInt` on line 6 work properly and make a successful call? If so, briefly explain why, if not, then say what happens.

(d) (3 points) Is the call on line 14 in the declarative kernel language? If not, briefly explain why it is not.

(e) (3 points) Suppose Oz used dynamic scoping. In that case, would the calls on lines 14 and 6 both be successful? If so, briefly explain why, if not, then say what would happen.

6. (15 points) [Concepts] Desugar the following Oz code into kernel syntax by expanding all syntactic sugars. (Assume that the identifier `Result`, and the function identifiers `SumTo` and `SumIter` are declared elsewhere.)

```
fun {SumTo N} {SumIter N 0} end
Result = {SumTo 7}
```

7. (10 points) [Concepts] What happens when the following code executes in Oz? Briefly explain why that happens.

```
local SetQ Q in
   Q = 4020
   SetQ = proc {$ V}
             Q = V
          end
   {SetQ 99}
   {Browse 'Q is '#Q}
end
```

8. (10 points) [Concepts] What is the output, if any, of the following code in Oz? Briefly explain why that output appears.

```
local Nat V in
   Nat = node(num: int(zero) color: orange value: 0)
   V = 444
   case Nat of
      vertex(number: N color: C value: V) then {Browse first#N#C#V}
   [] node(num: N color: C value: V) then {Browse second#N#C#V}
   [] node(num: int(M) color: C value: V) then {Browse third#M#C#V}
   [] node(num: N value: V) then {Browse fourth#N#V}
   else {Browse none(V)}
   end
end
```

9. [Concepts] Both Java and C# recently expanded by adding enhanced **for** loops. These are defined by telling programmers that use of such an enhanced **for** loop expands into an iterator call, a **while** statement, the given loop body, and another call to the iterator. The expanded version of a **for** loop is thus a statement that would be legal in older versions of the language.

   (a) (5 points) What is the term for this concept?

   (b) (10 points) Briefly describe one advantage of extending a language in this way.