Fall, 2010 Name: ____________________

COP 4020 — Programming Languages I

Test on the Declarative Model

Special Directions for this Test

This test has 13 questions and pages numbered 1 through 6.
This test is open book and notes, but no electronics.
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	10	11	12	13	Total
Points:	10	5	5	5	5	5	5	5	5	15	10	10	15	100
Score:														

1. (10 points) [Concepts]

   Consider the execution of the following Oz code.

```
local Square in
   local N in
      N = 10
      Square = proc {$ X ?R}
                  {Number.'*' X X R}
               end
      {Square N N}
      {Browse 'N is '#N}
   end
end
```

   Circle the letter that best describes what happens during the above code's execution.

   A. The browser displays `'N is'#10`, because `N` can only be assigned once.

   B. The browser displays `'N is'#100`, because 100 is the square of 10.

   C. This produces a failure (a "tell" exception is thrown).

   D. Execution of the code suspends.

   E. This produces a failure, and then afterwards the browser displays `'N is'#100`.

   F. The code does not compile, due to static scoping.

2. (5 points) [Concepts]

   Circle the letter of the true statement below.

   A. Static type checking is easy to implement in a language with dynamic scoping.

   B. Static type checking is impossible, in general, in a language with dynamic scoping.

   C. A language with dynamic type checking will execute more quickly at runtime than a language with static type checking.

   D. Too much syntactic sugar will rot your teeth.

3. (5 points) [Concepts]

   Circle the letter of the true and accurate statement below.

   A. A closure in programming is something that always finishes execution.

   B. A closure consists of code and an environment that remembers the free variable identifiers in the code.

   C. A closure consists of code and an environment that remembers the bound variable identifiers in the code.

   D. A closure cannot depend on any identifiers that it does not declare itself.

4. (5 points) [Concepts]

   Circle the letter of the true statement below.

   A. Static scoping is useful because it allows the meaning of code to be understood by looking only in the area of the program surrounding a given piece of code, before the program is run.

   B. Static scoping is useless because no one should have to understand programs before they are run, instead they should just try out the program and see what happens, as that is the only true test of a program's meaning.

   C. Dynamic scoping is useful because it allows curried functions to be defined and used easily, without any fuss about making closures.

   D. Dynamic scoping makes programs easy to understand, because it makes it very clear what will happen in a program before it is run.

5. (5 points) [Concepts]

   Circle the letter of the true statement below.

   A. A dataflow variable in Oz can be assigned any number of values at any number different times during a program's execution.

   B. A dataflow variable in Oz can only hold primitive values, such as integers and Booleans; that is, it cannot hold records.

   C. A dataflow variable in Oz can be assigned only once during a program's execution, and my be assigned any type of value.

   D. In Oz one cannot assign a dataflow variable to itself.

6. (5 points) [Concepts] [MapToLanguages] Consider the following Oz code:

```
local F in
   F = proc {$ Y ?R}
          {Number.'*' X Y R}
       end
   local Three in
      Three = 3
      local Res in
         {F Three Res}
         {Browse Res}
      end
   end
end
```

   Circle the letter of the true and accurate statement below.

   A. In the code shown above, X occurs as a free variabile identifier, and if its type at runtime is Int, then the code will execute without type errors.

   B. In the code shown above, X occurs as a bound variabile identifier, and if its type at runtime is Int, then the code will execute without type errors.

   C. In the code shown above, X occurs as a free variabile identifier, and if its declared type is Int, then the code will statically type check without type errors.

   D. In the code shown above, X occurs as a bound variabile identifier, and if its declared type Int, then the code will statically type check without type errors.

7. (5 points) The following refers to the code above. Circle the letter of the true and accurate statement below.

   A. The code shown above can be type checked statically, because Oz has static scoping.

   B. The code shown above can only be type checked dynamically, because Oz uses dynamic scoping.

   C. The code shown above can only be type checked dynamically, because Oz does not have a static type system with statically declared types.

8. [Concepts]

   (a) (2 points) In Oz's declarative computation model, is it possible to write a random number function, which takes no arguments and returns different results each time it is called? (Circle the letter of the true choice below.)

      A. No.

      B. Yes.

   (b) (3 points) Give a brief explanation for your answer above.

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 $\{\}$.

Also, recall that **declare** is *not* in the declarative kernel, so you should *not* imagine an implicit **declare** in the examples given for these problems.

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

```
local Res in
   local Q in
      {Append Ls Elems Q}
      Res = Q
   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.

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

```
FilterIter = proc {$ Ls Pred Acc R}
                case Ls of
                   '|'(1:E 2:Es) then local CompRes in
                                         {Pred E CompRes}
                                         if CompRes
                                         then local NewAcc in
                                                 NewAcc = '|'(1: E 2: Acc)
                                                 {FilterIter Es Pred NewAcc R}
                                              end
                                         else
                                            {FilterIter Es Pred Acc R}
                                         end
                                      end
                else {Reverse Acc R}
                end
             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.

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

```
if (close(a[i])) {
    float temp;
    temp = b[j];
    b[j] = a[i];
    a[i] = temp;
}
```

(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.

12. (10 points) [Concepts]

What is the output, if any, of the following Oz code?

```
local Text = sentence(np(det(the) adj(quick) noun(fox))
                      vp(verb(jumped over)
                         object(np(det(the) adj(old) noun(dog)))))
in
   case Text of
      sentence(np(D A N)) then {Browse first#D#A#N}
   [] sentence(vp(V O)) then {Browse second#V#O}
   [] sentence(np(D _ noun(N)) vp(V _)) then {Browse third#D#N#V}
   else {Browse none}
   end
end
```

13. [Concepts] Consider the following Oz code in Oz.

```
local Swap in
   local X=clear Y=water in
      fun {Swap Rec}
         case Rec of
            pair(X Y) then pair(Y X)
         else oops
         end
      end
      {Browse {Swap pair(blanket bingo)}}
   end
end
```

(a) (5 points) When the above code runs in Oz, what will happen? (Note: although `Browse` is not part of the declarative model, we are using it in this problem to simplify the problem's statement.)

(b) (10 points) Desugar the above code. That is, translate the above code into kernel syntax by expanding all syntactic sugars, so that your answer is an equivalent statement in the declarative kernel language.