Spring, 2008 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 5.

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

Problem	Points	Score
1	15	
2	5	
3	5	
4	15	
5	10	
6	10	
7	10	
8	15	
9	15	

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

```
M = proc {$ Ls F G ?R}
       case Ls of
          cons(hd: H tl: T) then local FRes in
                                     {F H FRes}
                                     local TailRes in
                                        R = cons(hd: FRes tl: TailRes)
                                        {M T F N TailRes}
                                     end
                                  end
       else R = nil
       end
    end
```

   (a) (5 points) [Concepts] Write, below, in set brackets, the entire set of the variable identifiers that occur free in the statement above. For example, write $\{V, W\}$ if the variable identifiers that occur free are V and W. If there are no variable identifiers that occur free, write $\{\}$.

   (b) (10 points) [Concepts] Write, below, in set brackets, the entire set of the variable identifiers that occur bound in the statement above. For example, write $\{V, W\}$ if the variable identifiers that occur bound are V and W. If there are no variable identifiers that occur bound, write $\{\}$.

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

```
public int run(int x, int y) {
    q = x+3;
    return q;
}
```

   (a) (2 points) Write below, in set brackets, the entire set of identifiers that occur free in the Java code above.

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

3. (5 points) [Concepts] In Oz, a closure stores an environment that remembers the values of variables that occur in the body of a procedure. Does it store values for the identifiers that occur free or for those that occur bound?

4. (15 points) [Concepts] Desugar the following Oz code into kernel syntax by expanding all syntactic sugars. (Assume that `Plus` and `Y` are declared elsewhere.)

```
fun {Comp F G X}
   Z = {Plus X Y}
in
   {F {G Z}}
end
```

5. (10 points) [Concepts] Which of the following are correct statements about scoping in Oz. (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) Oz uses dynamic scoping for identifiers, which is why Oz also uses dynamic type checking.

   (b) Oz uses dynamic scoping for identifiers, which makes it easy to predict the values of identifiers.

   (c) Oz uses static scoping for identifiers, which makes it impossible to do static type checking.

   (d) Oz uses static scoping for identifiers, which makes it possible to statically predict the types of identifiers.

   (e) Oz uses static scoping for identifiers, which requires it to make closures for procedure values.

6. (10 points) [Concepts] What happens when the following code executes in Oz? Briefly explain why that happens.

```
local Sum Total in
   Total = 0
   fun {Sum Ls}
      case Ls of
         nil then Total
      [] H|T then Total=Total+H
                  {Sum T}
      end
   end
   {Show {Sum 1|2|3|nil}}
end
```

7. (10 points) [Concepts] What is the output, if any, of the following code in Oz? Briefly explain why that output appears.

```
local P Unk W in
   P = person(height: 62 weight: 190 age: Unk)
   W = 55
   case P of
      building(height: H weight: W age: A) then {Browse first#H#W#A}
   [] person(height: H weight: 190) then {Browse second#H}
   [] person(height: H weight: W age: A) then {Browse third#H#W#A}
   [] person(height: H weight: X age: A) then {Browse fourth#H#X#A}
   else {Browse none}
   end
end
```

8. (15 points) [Concepts] Which of the following are correct statements about syntactic sugars and linguistic abstractions. (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) Programmers will often use syntactic sugars and linguistic abstractions, because they make it easier to write programs.

   (b) Programmers will avoid using syntactic sugars and linguistic abstractions, because they are always strictly less efficient than writing out the desugared form.

   (c) Language designers use syntactic sugars and linguistic abstractions as an easy way to explain and implement complex, but helpful, syntax.

   (d) Language designers use syntactic sugars and linguistic abstractions to confuse and baffle programmers with complex and useless syntax.

   (e) Syntactic sugars and linguistic abstractions are a bad way to extend a programming language, since they lead to complications that programmers do not need.

9. (15 points) [Concepts] Which of the following are correct statements about exception handling. (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) Good programmers do not need exception handling mechanisms because good programmers never make mistakes that cause exceptions.

   (b) Languages need an exception handling mechanism so programmers can avoid cluttering their code with tests of status codes or condition codes.

   (c) Throwing an exception immediately terminates the running of the entire program and cannot be stopped under program control.

   (d) Throwing an exception starts a search for an exception handler, which can control what happens when some exception is thrown.

   (e) When a **`try`** statement has a **`finally`** clause, the **`finally`** clause's body is only executed when the body of the **`try`** statement throws an exception.