Spring, 2008 Name: ______________________________

COP 4020 — Programming Languages 1

Test on the Demand-Driven Concurrent Model, Message Passing, the Relational Model, and Programming Models vs. Problems

Special Directions for this Test

This test has 6 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 in the demand-driven concurrent, message passing, or relational model (as in chapters 4, 5, and 9 of our textbook). The problem will say which model is appropriate. However, you must not use imperative features (such as cells and assignment) or the library functions `IsDet` and `IsFree`. But please use all linguistic abstractions and syntactic sugars that are helpful.

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. (This means you can use functions in the Oz base environment such as `Map`, `FoldR`, `Filter`, `Append`, `Max`, etc.) In the message passing model you can use `NewPortObject` and `NewPortObject2` as if they were built-in, and in the relational model you can use `Solve` as if it was built-in.

For Grading

Problem	Points	Score
1	10	
2	10	
3	10	
4	20	
5	30	
6	20	

1. (10 points) [UseModels]

   Using Oz's demand-driven concurrent model, write a lazy function

   ```
   TempDiff: <fun lazy {$ <IStream Int> Int}: <IStream Int>
   ```

   that takes an infinite stream of integers `Readings` and an integer `SetPoint` and returns a lazy infinite stream of differences, such that the i^{th} element of the result stream is the difference between the i^{th} element of `Readings` and the value of `SetPoint`. Thus when the i^{th} output element is negative, it means that the i^{th} element of `Readings` is less than `SetPoint`.

   The following are some examples, written using the `Test` function as in the homework.

   ```
   declare
   % UpFrom is a helper for testing purposes
   fun lazy {UpFrom N} N|{UpFrom N+1} end

   {Test {List.take {TempDiff {UpFrom 0} 25} 7}
    '==' [~25 ~24 ~23 ~22 ~21 ~20 ~19]}
   {Test {List.take {TempDiff {UpFrom 23} 25} 7}
    '==' [~2 ~1 0 1 2 3 4]}
   {Test {List.take {TempDiff {Append [5 30 26 26] {UpFrom 23}} 25} 7}
    '==' [~20 5 1 1 ~2 ~1 0]}
   ```

2. (10 points) [Concepts]

   (a) Does the function `TempDiff` in problem 1 need to be lazy? (Answer "yes" or "no.")

   (b) Briefly explain your answer, using as an example the testing code problem 1.

3. (10 points) [Concepts]

   Consider the following Oz expression.

   ```
   {MyFun Arg1} == {MyFun Arg1}
   ```

   For which of the following computation models could the above expression both terminate normally and have the value **`false`**? (Circle the letters next to all those for which a result of false is possible. Don't circle the letters of the others. You are supposed to know what these models are.)

   (a) Declarative
   (b) Declarative Concurrent
   (c) Demand-Driven Concurrent
   (d) Message Passing
   (e) Relational

4. (20 points) [UseModels]

   Using Oz's demand-driven concurrent model, write a function

   ```
   HeatControl: <fun {$ <IStream Int> Int}: <IStream Int>
   ```

   that takes an infinite stream of integers `Diffs` and a positive integer `Slop` and returns a lazy infinite stream of non-negative integers. The integers in the output can be thought of as directives of how much heat to generate, with 0 meaning no heat (i.e., the heater is off). The heater, which might be used for a swimming pool, starts being off. When the heater is off, the i^{th} element of the output stream is 0 unless the i^{th} element of `Diffs`, D_i, is strictly less than $-\texttt{Slop}$, and then the output is the minimum of $\lfloor -D_i/2 \rfloor$ and 10. When the heater is on and D_i is negative, then the i^{th} output is the minimum of $\lfloor -D_i/2 \rfloor$ and 10. When the heater is on and D_i is non-negative, then the i^{th} output is 0.

   You should assume that `Slop` is strictly greater than 0. Don't write any extra code or repeated code.

   Hints: Recall that in Oz, `D div 2` computes $\lfloor \texttt{D}/2 \rfloor$.

   The following are some examples, written using the `Test` function as in the homework.

```
declare
% The following are helper functions, for testing purposes
fun lazy {DownFrom N} N|{DownFrom N-1} end
fun lazy {Steady N} N|{Steady N} end

{Test {List.take {HeatControl {DownFrom 0} 3} 8} '==' [0 0 0 0 2 2 3 3]}
{Test {List.take {HeatControl {Steady ~2} 3} 7} '==' [0 0 0 0 0 0 0]}
{Test {List.take {HeatControl {Steady ~3} 3} 7} '==' [0 0 0 0 0 0 0]}
{Test {List.take {HeatControl {Steady ~4} 3} 7} '==' [2 2 2 2 2 2 2]}
{Test {List.take {HeatControl {DownFrom ~18} 3} 7} '==' [9 9 10 10 10 10 10]}
{Test {List.take {HeatControl {Append [0 ~3 ~2 ~3 ~4 ~2 ~3] {DownFrom ~5}} 3} 8}
 '==' [0 0 0 0 2 1 1 2]}
{Test {List.take {HeatControl {DownFrom 3} 4} 11}
 '==' [0 0 0 0 0 0 0 0 2 3 3]}
{Test {List.take {HeatControl {Append [5 30 26 26] {DownFrom 2}} 1} 11}
 '==' [0 0 0 0 0 0 0 0 1 1 2]}
```

5. (30 points) [UseModels]

Using Oz's Message Passing model, write a function `NewThermostat` that takes a port object `Heater` as an argument, and returns a port object. The returned port object acts as a thermostat that controls the heater by sending it messages of the form `run(`N`)`, where N is a number between 0 and 10 (inclusive), with level 0 being "off" and level 10 being maximum heat.

The thermostat port object responds to the following messages:

- `currentReading(`T`)`, where T is an integer, which tells the thermostat that the temperature of the system is T degrees.
- `setPoint(`S`)`, where S is an integer, which tells the thermostat the user's wish for the system's desired temperature.

When created the thermostat first sends a `run(0)` message to the heater port object. This makes the heater's level (N) start at 0. The initial set point (desired temperature, S) of the system is 80 degrees. We also assume that the current temperature (T) at the start is also 80 degrees.

Following this initialization the thermostat responds to each message it receives by sending a message to the heater. Suppose the heater is currently set to run at level N, the current set point is S, and the current system temperature is T.

If a message `currentReading(`T'`)` arrives, then the thermostat sends the message `run(`N'`)` to the heater, where N' is 0 if N is 0 and $-2 \leq T' - S$, and otherwise (if either N is nonzero and $T' - S < -2$) it makes N' be the minimum of $\lfloor -(T' - S)/2 \rfloor$ and 10.

If a message `setPoint(`S'`)` arrives, then the thermostat sends the message `run(`N'`)` to the heater, where N' is 0 if N is 0 and $-2 \leq T - S'$, and otherwise (if either N is nonzero and $T - S' < -2$) it makes N' be the minimum of $\lfloor -(T - S')/2 \rfloor$ and 10.

Also, whenever a message is received the current heater level, system temperature, and set point are appropriately updated.

You should assume that no other agents send messages to the heater object; thus the heater object passed as the argument to the `NewThermostat` function is "owned" by that function.

Hints: Recall that in Oz, `D div 2` computes $\lfloor \text{D}/2 \rfloor$. Use `NewPortObject`.

The following are some examples, written using the `Test` function as in the homework.

```
declare
local Commands Heater Thermo in
   Heater = {NewPort Commands}
   Thermo = {NewThermostat Heater}
   {Test {List.take Commands 1} '==' [run(0)]}
   {Send Thermo currentReading(78)}
   {Test {List.take Commands 2} '==' [run(0) run(0)]}
   {Send Thermo currentReading(77)}
   {Test {List.take Commands 3} '==' [run(0) run(0) run(1)]}
   {Send Thermo currentReading(70)}
   {Test {Nth Commands 4} '==' run(5)}
   {Send Thermo setPoint(72)}
   {Test {Nth Commands 5} '==' run(1)}
   {Send Thermo currentReading(68)}
   {Test {Nth Commands 6} '==' run(2)}
   {Send Thermo currentReading(71)}
   {Test {Nth Commands 7} '==' run(0)}
   {Send Thermo currentReading(70)}
   {Test {Nth Commands 8} '==' run(0)}
end
% More examples on the next page...
```

```
local Commands Heater Thermo in
   Heater = {NewPort Commands}
   Thermo = {NewThermostat Heater}
   {Test {Nth Commands 1} '==' run(0)}
   {Send Thermo setPoint(82)}
   {Test {Nth Commands 2} '==' run(0)}
   {Send Thermo setPoint(83)}
   {Test {Nth Commands 3} '==' run(1)}
   {Send Thermo setPoint(84)}
   {Test {Nth Commands 4} '==' run(2)}
   {Send Thermo currentReading(77)}
   {Test {Nth Commands 5} '==' run(3)}
   {Send Thermo currentReading(54)}
   {Test {Nth Commands 6} '==' run(10)}
   {Send Thermo currentReading(64)}
   {Test {Nth Commands 7} '==' run(10)}
   {Send Thermo setPoint(70)}
   {Test {Nth Commands 8} '==' run(3)}
   {Send Thermo currentReading(69)}
   {Test {Nth Commands 9} '==' run(0)}
   {Send Thermo setPoint(71)}
   {Test {Nth Commands 10} '==' run(0)}
end
```

6. (20 points) [EvaluateModels] For each of the following programming problems, (i) name the best programming model for solving the problem, and (ii) briefly explain why that model is best for the problem. When in doubt, favor the least expressive model (i.e., the one with the fewest features) that can solve the problem. (Choose from among the programming models we studied this semester.)

   (a) A server that allows several different users to exchange cooking recipes. Each recipe is a record containing a title, a list of ingredients, and a list of cooking steps. Users may submit a new recipes or retrieve recipes by using its name or ingredients. Users may be submitting and looking up recipes concurrently, but the server only has to handle one request at a time.

   (b) A function that manipulates web pages written in XHTML (which is tree-structured) to produce a similar web page but with all images removed. This function would be used by a cell phone browser, in a mode where the user only wants to display text.

   (c) A function that can find good schedules for what classes computer science students should take, given the degree requirements, what times the classes are offered, what classes they have already taken, and what their work schedule is. No program like this currently exists, and so you are not sure what the customer really wants such a program to do. Thus it's important to show the customer a prototype quickly.

   (d) A library of combinatorial logic gates, such as "and" and "or" gates, that can be instantiated and connected in various ways, with delays. For example, the user can use this library to simulate latches and other small circuits by combining instances of these gates, and observe the results for various inputs. It's important that you develop this library in a way that minimizes your overall programming time.