Fall, 2006 Name: _____

Com S 541 — Programming Languages 1

Test on the Declarative Concurrent and Message Passing Models

Special Directions for this Test

This test has 4 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 2–5 of our textbook. But unless specifically directed, you should not use cells (explicit state).

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	25	
2	30	
3	25	
4	20	

1. (25 points) In this problem we will examine the advantages and disadvantages of using different programming models for writing a function, `Random`, that is supposed to generate random integers. The type of `Random` is supposed to be `<fun {$}: <Int>>`. While loosely specified, `Random` must *not* return the same integer every time it is called.

   (a) Is it possible to correctly implement `Random` using just features of the declarative model? If so, sketch a solution; if not, briefly explain why not.

   (b) Is it possible to correctly implement `Random` using just features of the declarative concurrent model? If so, sketch a solution; if not, briefly explain why not.

   (c) Is it possible to correctly implement `Random` using just features of the message passing model? If so, sketch a solution; if not, briefly explain why not.

2. (30 points) Using the message passing model, implement the a data abstraction "Box", with the following operations.

```
NewBox: <fun <$ Value>: Box>
ExchangeBox: <proc <$ Box Value Value>>
AssignBox: <proc <$ Box Value>>
AccessBox: <fun <$ Box>: Value>
```

A Box is like a Cell, in that it holds a value (of any type). The function call `{NewBox X}` returns a new Box containing the value `X`. The procedure call `{ExchangeBox B Old New}` atomically binds `Old` to the value in box `B` and makes `New` be the new value contained in Box `B`. The procedure call `{AssignBox B V}` makes the value `V` be the new value of Box `B`. The function call `{AccessBox B}` returns the value contained in Box `B`.

The code should pass the following tests (which use the test harness from the homework).

```
\insert 'Test.oz'
declare B1 B2 V1old V2old
{StartTesting 'Box operations'}
B1 = {NewBox 1}
B2 = {NewBox 2}
{ExchangeBox B1 V1old 7}
{ExchangeBox B2 V2old 99}
{Test V1old '==' 1}
{Test V2old '==' 2}
declare V1x V2x
{ExchangeBox B1 V1x 88}
{ExchangeBox B2 V2x 333}
{Test V2x '==' 99}
{Test V1x '==' 7}
{Test {AccessBox B1} '==' 88}
{Test {AccessBox B2} '==' 333}
{Test {AccessBox B2} '==' 333}
{Test {AccessBox B1} '==' 88}
{AssignBox B1 4}
{Test {AccessBox B1} '==' 4}
{Test {AccessBox B2} '==' 333}
{AssignBox B1 asymbolliteral}
{Test {AccessBox B1} '==' asymbolliteral}
declare
X=1 Y=2 Z=3
B={NewBox Z}
{StartTesting 'Some equations'}
{Test {AccessBox {NewBox X}} '==' X}
{AssignBox B Y}
{Test {AccessBox B} '==' Y}
```

You can use the following file, `NewObject.oz`, in your solution.

```
% $RCSfile: NewPortObject.oz,v $
declare
fun {NewPortObject Init Fun}
   Sin Sout
in
   thread {FoldL Sin Fun Init Sout} end
   {NewPort Sin}
end
```

You are *not* allowed to use cells in your solution!

There is space for your answer on the next page.

Please write your solution by filling in the following outline.

Note that this outline requires you to write `AssignBox` and `AccessBox` using the `ExchangeBox` procedure.

```
\insert 'NewPortObject.oz'
declare
fun {NewBox X}









end

proc {ExchangeBox B ?Old ?New}



end

proc {AssignBox B X}
   {ExchangeBox               }


end

fun {AccessBox B}
   Val
in
   {ExchangeBox               }


end
```

3. This is a problem about programming in the declarative concurrent model (i.e., without using `NewPort` or `Send`), and about composing pipelines that reuse functions on streams.

   Your solution for each of the following parts must produce its outputs incrementally.

   (a) (5 points) Write a function `Ints`, such that the call `{Ints N}` produces an infinite stream of integers whose ith element is i+`N`. The following are tests.

   ```
   \insert 'Test.oz'
   declare
   {Test {List.take {Ints 1} 4} '==' [1 2 3 4]}
   {Test {List.take {Ints 10} 6} '==' [10 11 12 13 14 15]}
   ```

   Please write your answer below.

   (b) (10 points) Reusing your function `Ints`, write a function `Squares`, such that the call `{Squares}` produces an infinite stream of integers, whose nth element is n^2.

   ```
   \insert 'Test.oz'
   declare
   {Test {List.take {Squares} 4} '==' [1 4 9 16]}
   {Test {List.take {Squares} 6} '==' [1 4 9 16 25 36]}
   ```

   Please write your answer below.

   ```
   \insert 'Ints.oz'  % answer to the previous part
   ```

(c) (10 points) Reusing your function `Squares`, write a function `SquaresDigits`, such that the call `{SquaresDigits}` produces an infinite stream of lists of integers, whose nth elements is the list of digits (in base 10) of the numbers in n^2.

```
\insert 'Test.oz'
declare
{Test {List.take {SquaresDigits} 4}
 '==' [[1] [4] [9] [1 6]]}
{Test {List.take {SquaresDigits} 11}
 '==' [[1] [4] [9] [1 6] [2 5] [3 6] [4 9] [6 4] [8 1] [1 0 0] [1 2 1]]}
```

Please write your answer below. (You can use Oz's **`div`** and **`mod`** operators which do integer division and modulus, respectively.)

```
\insert 'Squares.oz' % answer to the previous part
```

4. This is a problem about comparing programming models.

   (a) (10 points) Problem 3 above asks you to work without using the message passing primitives, and instead to use the declarative concurrent model. Would it have been easier to use the message passing model to solve that problem? Briefly explain your answer.

   (b) (10 points) Suppose you are asked to program a simulation of an agent-based auction system for someone doing research in economics. This system consists of several independent agents, each of which must communicate with a central auction server to evaluate merchandise, place bids, and make payments.

   Among the programming models we studied this semester, What is the most restrictive (i.e., the least expressive or smallest) programming model that can practically be used program the overall structure of such a system? Justify your answer.