Name: _____

COP 4020 — Programming Languages I (Spring 2011)

Test on Declarative Concurrency

Directions for this test

Please read these instructions carefully before starting.

This test has 7 questions and pages numbered 1 through 9. This test is open book and notes, but no electronics.

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 1–4 of our textbook. But unless specifically directed, you should not use imperative features (such as cells) or the library functions `IsDet` and `IsFree`.

You are encouraged to use and/or 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 their definition on your test. You can use the built-in functions in the Oz base environment such as `Append`, `Not`, `Reverse`, `Length`, `Filter`, `Map`, `FoldL`, `Member`, `FoldR`, `List.take`, `List.drop`, etc.

Recall that the grammar for the type of infinite streams with elements of type `T` is as follows.

⟨IStream T⟩ ::= `T '|'` ⟨IStream T⟩

For Grading

Question:	1	2	3	4	5	6	7	Total
Points:	10	10	10	15	20	15	20	100
Score:								

1. (10 points) [Concepts]

   Consider the following code and select the correct answer below:

```
% $Id: p1.oz,v 1.1 2011/04/24 15:32:59 fhussain Exp $

local X Y Z P in
   thread Z = Y*Y-1 end
   thread Y = X + 1 end
   thread if Y == P then skip else skip end end
   thread P = 3 end
   thread X = 10 end
   {Wait Z}
   {Show [X Y Z P]}
end
```

   A. When the code is executed, it results in an exception due to a unification failure error.

   B. When the code is executed the program suspends.

   C. The behavior of this program cannot be predicted due observable nondeterminism.

   D. When the code is executed, it terminates normally with the emulator window displaying '[10 11 120 3]'.

2. (10 points) [Concepts]

   Consider the following program and select the correct answer below:

```
% $Id: p2.oz,v 1.1 2011/04/24 15:32:59 fhussain Exp $

local X Y Ans in
   thread {ByNeed proc {$ K} K = 50 end X} end
   thread X = Y end
   Ans = thread if X ==  Y then true else false end end
   {Wait Ans}
   {Show [X Y Ans]}
end
```

   A. The value '[50 50 true]' is displayed in the emulator window and the program terminates normally.

   B. The program terminates normally with the emulator window displaying placeholders (like _, _<optimized> or _<simple>) for X and Y and the value true for Ans.

   C. The program suspends due to the blocked statement in the third thread.

   D. The program throws an exception due to a unification failure error.

3. (10 points) [Concepts] Consider the following code and choose the correct answer below:

```
% $Id: p3.oz,v 1.1 2011/04/24 15:32:59 fhussain Exp $

local A B C in
   thread  if A == B then C = true else C = false end end
   thread A = 10 end
   {Wait C}
   {Show [A B C]}
end
```

   A. On execution, the program suspends.

   B. On execution the program terminates normally with the emulator window displaying placeholders for `A`, `B` and `C`.

   C. The program terminates normally with the emulator window displaying [10 `<placeholder>` false].

   D. The program throws an exception due to a unification failure error.

4. [EvaluateModels] Which is the least expressive model that can solve the following problems. Briefly justify your answer.

   (a) (5 points) A function that takes an infinite stream of increasingly more accurate approximations to the real cubic root of a floating point number, allowing the caller to find the approximation that precise enough according to some user specified criteria.

   (b) (5 points) A function that takes a list of employee records containing information about his health insurance history and financial compensation data as argument and returns a new list with the data updated for the current year, while leaving old data unchanged.

   (c) (5 points) A word-count function in a text editor program that waits on an input stream and calculates the current word count every time new words arrive on the stream.

5. [Concepts][UseModels]

Consider the code below and answer the questions that follow:

```
declare
fun lazy {Anon X Y Z}
   if X == Y then Z+1 else Z-1 end
end
```

(a) (5 points) What is the output when the function definition above and the following code are fed to Oz?

```
local Ans in
   Ans = local Retval Arg1 Arg2 Arg3 in
            Arg1 = 10
            Arg2 = 12
            Arg3 = 17
            Retval = {Anon  Arg1 Arg2 Arg3}
            Retval
         end
   {Browse Ans}
end
```

(b) (5 points) What is the output when the function definition above and the following code are fed to Oz?

```
local SquaredAns in
   SquaredAns = local Retval Arg1 Arg2 Arg3 in
            Arg1 = 10
            Arg2 = 12
            Arg3 = 17
            Retval = {Anon  Arg1 Arg2 Arg3}
            Retval*Retval
         end
    {Wait SquaredAns}
    {Browse SquaredAns}
end
```

(c) (5 points) What is the output when the function definition above and the following code are fed to Oz?

```
local SquaredAns in
   SquaredAns = local Retval Arg1 Arg2 Arg3 in
            Arg1 = 10
            Arg2 = 12
            Arg3 = 17
            Retval = {Anon  Arg1 Arg2 Arg3}
            Retval*Retval
         end
   SquaredAns = 256
   {Browse SquaredAns}
end
```

(d) (5 points) What is the output when the function definition above and the following code are fed to Oz?

```
local SquaredAns in
   SquaredAns = local Retval Arg1 Arg2 Arg3 in
            Arg1 = 10
            Arg2 = 12
            Arg3 = 17
            Retval = {Anon  Arg1 Arg2 Arg3}
            Retval*Retval
         end
   SquaredAns = 16
   {Browse SquaredAns}
end
```

6. (15 points) [UseModels] Using Oz's demand-driven concurrent model, write a function

```
HoppingBlocks: <fun lazy {$ <IStream T> <Int> <Int>}: <IStream <List T>>
```

that takes an infinite stream, `Istrm` of elements of some type `T`, a strictly positive integer `BlkSz` and a non-negative integer `SkipSz`, and produces a stream of lists of type `T` such that the the first element in the resultant stream is a list of the first `BlkSz` elements of `Istream`; the next `SkipSz` elements of `IStream` are ignored; then the second element of the resultant stream is a list of the next `BlkSz` elements and so on. The following are some examples:

```
% $Id: HoppingBlocksTests.oz,v 1.3 2011/04/25 04:05:34 fhussain Exp $
\insert 'TestingNoStop.oz'
\insert 'HoppingBlocks.oz'

declare
fun lazy {From M} M|{From M+1} end


{StartTesting 'HoppingBlock $Revision: 1.3 $'}
{Test {List.take {HoppingBlocks {From 100} 4 2} 5} '=='
 [[100 101 102 103] [106 107 108 109] [112 113 114 115] [118 119 120 121] [124 125 126 127]]}
{Test {List.take {HoppingBlocks {From 1} 3 10} 4} '==' [[1 2 3] [14 15 16] [27 28 29] [40 41 42]]}
{Test {List.take {HoppingBlocks {From 10} 1 0} 10} '==' [ [10] [11] [12] [13] [14] [15] [16] [17] [18] [19]]}
{EndTesting 'done'}
```

7. (20 points) [UseModels] Using Oz's declarative concurrent model, write a function:

```
StreamFiltering: <fun lazy {$ <IStream T> <Int> <Int> <fun {$ T}: Boolean>}: <IStream <List T>>
```

that takes an infinite stream `Istream` of elements of some type `T`, a strictly positive integer `BlkSz`, a non-negative integer `SkipSz` and and a unary predicate `Pred` and returns a stream of lists, `ResStrm`.

`ResStrm` consists of lists of size `BlkSz`, where each list comprises consecutive elments of type `T` taken from `IStream` such that least one element of each list satisfies `Pred`. Like the previous question, we skip `SkipSz` elements from `Istream` each time we extract `BlkSz` elements to be considered for the output stream. The following are examples.

```
% $Id: StreamFilteringTests.oz,v 1.4 2011/04/25 16:10:55 fhussain Exp $

\insert 'TestingNoStop.oz'
\insert 'StreamFiltering.oz'

fun {BStream}
   fun lazy {AlternateBStream N}
      if (N mod 2) == 0 then true else false end|{AlternateBStream N+1}
   end
in
   {AlternateBStream 0}
end

fun lazy {FromBy X B} X|{FromBy X+B B} end

{StartTesting 'StreamFiltering $Revision: 1.4 $'}
{Test {List.take {StreamFiltering {FromBy 10 3} 2 5 fun {$ X} {IsEven X} end} 3} '==' [[10 13] [31 34] [52 55]]}
{Test {List.take {StreamFiltering {FromBy 1 4} 4 20 fun {$ X} X<50 orelse X>300 end} 4}'=='
 [[1 5 9 13] [337 341 345 349] [393 397 401 405] [449 453 457 461]]}
{Test {List.take {StreamFiltering {FromBy 1 2} 5 10 fun {$ X} X<50 orelse X>150 end} 5} '=='
 [[1 3 5 7 9] [31 33 35 37 39] [151 153 155 157 159] [181 183 185 187 189] [211 213 215 217 219]]}
{Test {List.take {StreamFiltering {BStream} 4 1 fun {$ X} X == true end} 3} '=='
 [[true false true false] [false true false true] [true false true false]]}
{Test {List.take {StreamFiltering {BStream} 1 0 fun {$ X}  X == true end}  5} '=='
 [[true] [true] [true] [true] [true]]}
{EndTesting 'done'}
```

Hint: It maybe helpful to use a lazy version of one of the standard Oz list functions (§4.5.7).

Answer:

```
\insert 'HoppingBlocks.oz' %You can use the previous problem's answer without rewriting it here.
```