Ren Hao Wong
CSCI 117 Lab 6

Part 1

1.

```
S1 S2 S3 T1 T2
S1 S2 S3 T1 S3.1 T2 -- displays true
S1 S2 S3 T2 T1
S1 S2 S3 T2 S3.1 T1 -- no display

S1 S2 T1 T2
S1 S2 T1 S3 T2
S1 S2 T1 S3 S3.1 T2 -- displays true
S1 S2 T2 T1
S1 S2 T2 S3 T1
S1 S2 T2 S3 S3.1 T1 -- no display

S1 T1 S2 T2
S1 T1 S2 S3 T2
S1 T1 S2 S3 S3.1 T2 -- displays true
```

2.

<Terminal output (quantum of Infinity)>:

```
Y : Unbound

T2 : Unbound

T1 : Unbound
```

<Terminal output (quantum of 1)>:

```
Y : 3

T2 : 3

T1 : Unbound
```

When the quantum is infinite, the program completes a thread before moving to another. The unification between "Y" and "X" did not occur before Browsing the value of "Y", thus "Y" remained unbounded at that moment. Similarly, the binding of values for "T2" and "T1" occur in branching threads, and their binding happens only after Browsing "T2" and "T1", therefore their values are displayed as unbounded.

When the quantum is one, the program executes 1 statement on each thread while cycling between the threads to execute. This allows "Y" to be bounded to "X" whereas "X" is then bounded to the value of 3, before the Browse Y statement. Because the program executes strict one statement for each thread, "T2" can be bounded because its thread only contains a statement that is to bind its value to 3. "T1" however required performing an addition statement of 4 + 3 before the binding statement, adding to a total of 2 statements in its thread. While the addition statement may be performed, the binding statement could not be performed before the Browse T1 statement located on the main thread.

3.

<Terminal output>:

```
X : 1

X : 1

Y : 2

Z : 3

Z : 3

Z : 3

X : 1

X : 1

Y : 2

Y : 2

Y : 2

Z : 3

Z : 3

X : 1

Y : 2
```

<Code>

```
local Z in
  Z = 3
  thread local X in
    X = 1
    skip Browse X
    skip Browse X
    skip Basic

    skip Browse X
    skip Browse X
    skip Basic
    skip Basic

    skip Browse X
    end
  end
  thread local Y in
    Y = 2
    skip Browse Y
    skip Basic
    skip Basic

    skip Browse Y
    skip Browse Y
    skip Browse Y
    skip Basic

    skip Browse Y
    end
  end
  skip Browse Z
  skip Browse Z
  skip Browse Z
  skip Basic

  skip Browse Z
  skip Browse Z
 end
```

4.

<Terminal output>:

```
ghci> runFullT (Finite 5) "declarative threaded" "Lab6/thread.txt"
"Lab6/thread.out"
```

```
thread suspended: [(if EXU1 then [skip/BB] else
[skip],[("EXU1",9),("B",8),("IntPlus",1),("IntMinus",2),("Eq",3),("GT
",4),("LT",5),("Mod",6),("IntMultiply",7)])]
B : true()

ghci> runFullT (Finite 4) "declarative threaded" "Lab6/thread.txt"
"Lab6/thread.out"
B : true()
```

The minimum quantum that will cause a suspension to occur is 5.

`local B in`
`  B = thread true end`
`  if B then skip Browse B`
`end`
`end`
	S1:declaration of local B
	S2:declaration of thread
	T1:binding of B to true
	S3:declaration of local condition as variable
	S4:binding of condition to B
	S5:if-statement condition checking
	S6:skip Browse B

Execution sequence with a quantum of 5: S1 S2 S3 S4 S5 T1 S6

When the syntactic sugar is converted to kernel syntax, additional statements are added in between, thus requiring a minimum quantum of 5 to cause the suspension.

5.

The Fibonacci algorithm for this lab has a logic error on Fib(0) = 1 instead of 0, causing the output sequence to be left-shifted by one unit. The following tests were performed with a quantum of 3.

a.

X	Result	fib1_sugar	fib2_sugar	fib1_thread
8	34	0.06 secs, 25,609,312 bytes	0.02 secs, 4,483,280 bytes	0.13 secs, 61,035,176 bytes
9	55	0.10 secs, 56,089,664 bytes	0.02 secs, 4,723,808 bytes	0.28 secs, 149,729,752 bytes
10	89	0.22 secs, 131,054,088 bytes	0.01 secs, 4,980,584 bytes	0.70 secs, 377,477,976 bytes
11	144	0.50 secs, 319,037,784 bytes	0.02 secs, 5,247,224 bytes	1.78 secs, 964,527,896 bytes
12	233	1.23 secs, 797,279,928 bytes	0.02 secs, 5,525,568 bytes	4.64 secs, 2,482,193,824 bytes

13	377	3.15 secs, 2,025,800,392 bytes	0.01 secs, 5,816,288 bytes	12.08 secs, 6,412,957,008 bytes
14	610	8.81 secs, 5,202,536,168 bytes	0.03 secs, 6,119,976 bytes	31.06 secs, 16,607,098,440 bytes
15	987	20.55 secs, 13,452,712,296 bytes	0.02 secs, 6,435,096 bytes	82.93 secs, 43,069,404,368 bytes

fib2_sugar with the iterative algorithm is the fastest as well as the most memory efficient variant because the iteration does not exponentially increase the procedures in the call stack. fib1_sugar with the unthreaded recursive algorithm ranks second in terms of performance because its recursive calls causes a lot of instructions to be added to the call stack, as well as creating recursion branches on each cycle that is not a base case. Fib1_thread with the threaded recursion is the slowest and most memory inefficient, as a lot of threads were suspended and waiting for each other to complete in between the recursions.

b.

n	Result	Threads created	
		`(thread {Fib (In - 1)} end + thread {Fib (In - 2)} end)`	`(thread {Fib (In - 1)} end + {Fib (In - 2)})`
0	1	0	0
1	1	0	0
2	2	2	1
3	3	4	2
4	5	8	4
5	8	14	7
6	13	24	12
7	21	40	20
8	34	66	33

The first version of the recursive call creates twice as much threads as the second version. The pattern for the number of threads created for the second version is the sum of Fibonacci results from $n - 2$ to n_0

Example:

Fib (5): 7 threads created = $Fib(3) + Fib(2) + Fib(1) + Fib(0) = 3 + 2 + 1 + 1$

Part 2

```
local Producer OddFilter Consumer in

    Producer = proc {$ N Limit Out}
        if (N < (Limit + 1)) then T N1 in
            Out = (N|T)
            N1 = (N + 1)
            {Producer N1 Limit T}
        else
            Out = nil
        end
    end

    OddFilter = proc {$ P Out}
        case P
        of nil then
            Out = nil
        [] (X|Xs) then Filtered in
            {OddFilter Xs Filtered}

            if ((X mod 2) == 0) then
                Out = (X|Filtered)
            else
                Out = Filtered
            end
        end
    end

    Consumer = fun {$ P} in
        case P
        of nil then
            0
        [] (X|Xs) then
            (X + {Consumer Xs})
        end
    end

    // Example Testing
    local N L P F C in
        N = 0
```

```
        L = 100

        // [0 1 2 .. 100]
        thread
            {Producer N L P}
            skip Browse P
        end

        // [0 2 4 .. 100]
        thread
            {OddFilter P F}
            skip Browse F
        end

        thread C = {Consumer F}
            skip Browse C
        end
    end
end
```