Quiz 10a

1. (a) (4 points)

In weaving, a horizontal thread is pulled through a bunch of vertical threads. The horizontal thread passes over some of the vertical ones, and under others. The choice of over or under determines the pattern of the weave.

We will represent a pattern as a list of the words OVER and UNDER, repeated as needed. Here's an example:

(OVER OVER UNDER OVER UNDER UNDER)

The pattern may be of any length (it depends on the desired width of the woven cloth), but it must contain at least one OVER and at least one UNDER.

Write a Scheme expression to compute the (infinite) stream of all possible patterns.

**(define foo**

**(cons-stream '()**

**(interleave (stream-map (lambda (p) (cons 'over p)) foo)**

**(stream-map (lambda (p) (cons 'under p)) foo))))**

**(define patterns**

**(stream-filter (lambda (p) (and (member 'over p) (member 'under p)))**

**foo))**

**You can't combine these two steps; several people tried this:**

**(define patterns ;;; wrong!**

**(stream-filter (lambda ...)**

**(cons-stream '() (interleave (stream-map ... patterns)**

**(stream-map ... patterns)))))**

**3 points for this ^**

**Another wrong solution was to try to avoid the need for filtering by**

**"priming the pump" with initial patterns other than the empty one,**

**like this:**

**(define patterns ;;; wrong!**

**(cons-stream '(over under)**

**(cons-stream '(under over)**

**(interleave ...))))**

**2 points for this ^**

**1 point for something that has some idea.**

2. (3 points)

We want to generate the following infinite stream of infinite streams:

((1 2 3 4 ...) (2 3 4 5 ...) (3 4 5 6 ...) (4 5 6 7 ...) ...)

(Of course it doesn't look like that if you print it in STk.)

You are given the streams ones of all ones, and ints of all integers starting from one. You can use any stream procedures in the text, but don't define new procedures. Fill in the blanks:

(define stream-stream

(cons-stream **ints**

(**stream-map (lambda (x) (stream-map + ones x)) stream-stream)))**

1 point for starting stream with ints.

1 point for each of the maps.

3. (3 points)

Consider the following procedure:

(define (foo fn start)

(define result

(cons-stream start (stream-map fn result)))

result)

For each of the following streams, we would like to know whether

the stream can be created by a single invocation of foo.

If so, show the necessary invocation of foo including its

arguments. If not, just say no.

(a) The stream of integers (1 2 3 4 5 ...).

(foo 1 1+)

1 point

(b) The stream of powers of 2 (1 2 4 8 16 ...).

(foo 1 (lambda (x) (\* x 2)))

1 point

(c) The stream of Fibonacci numbers (0 1 1 2 3 5 8 13 ...).

No

1 point