Quiz 10b Rubric

1. (1+2 points) We can also use a cyclic list to represent the infinite list of ones:

(define ones (list 1))

(set-cdr! ones ones)

(list-ref ones x) will give you back 1 for any valid x, so this really does represent an infinite list of ones. In this question, we explore some reasons why streams are better than cyclic lists.

* 1. Although we can represent the infinite list of ones, we cannot represent the infinite list of integers as a cyclic list. Explain why.

In a cyclic list, we get the “infiniteness” by going back and repeating previous elements. In the list of integers though, every element is unique, and so cannot be easily represented by something which requires you to repeat previous elements.

1 point, probably all or nothing

* 1. Give an example of code using list functions that would cause an infinite loop when used with the cyclic representation, but the corresponding code using stream functions would work fine for an infinite stream of ones. For example, if you think that taking the car of the cyclic list would cause an infinite loop, but taking the stream-car of the stream of ones is fine, you should say (car ones). You may define helper procedures, but there are simple answers that do not require this. If your answer is wrong, you may get partial credit for an explanation of your answer.

(map 1+ ones) – since map never sees a null, it never hits the base case and terminates. However, stream-map would work.

2 points, partial credit on the basis of their explanations. Explanation is **not** required to get full credit.

1. (3 points) Here is an example of some very poorly written code – it depends on the implementation of streams (because it uses set!). As a result, for this question you should check and double-check your answers mechanically.

(define glob 1)

(define mystery

(cons-stream 1

(stream-map (lambda (x) (set! glob (+ glob 2))

(+ x glob))

mystery)))

After typing in these definitions, we execute the following code:

(stream-car (stream-cdr (stream-cdr mystery)))

(stream-car mystery)

* 1. What would the results be, assuming that the promises are memoized?

(stream-car (stream-cdr (stream-cdr mystery))) 9 (1 point)

(stream-car mystery) 1 (0.5 point)

* 1. What would the results be, assuming that the promises are not memoized? (Yes, there is a difference in the results of at least one of the expressions.)

(stream-car (stream-cdr (stream-cdr mystery))) 13 (1 point)

(stream-car mystery) 1 (0.5 points)

1. (3+1 points) Let’s write a procedure to find the smallest factor of a number, apart from 1. We will use streams for this procedure. Here’s the algorithm to find the smallest factor of n:

Return the first element in the stream of the factors of n, which is found by filtering the elements of the stream of numbers between 2 and n.

1. Implement the procedure, using the algorithm above. Don’t use infinite streams.

(define (smallest-factor n)

(stream-car (stream-filter (lambda (x) (= (remainder n x) 0))

(stream-enumerate-interval 2 n))))

Don’t take off points for swapping the arguments to remainder. Don’t take off points for using infinite streams – I just put that in so that they wouldn’t use that as an answer for b.

-0.5 points if solution uses numbers till sqrt(n) or n-1 (causes error for prime numbers).

-1 point if it returns a stream instead of a number

-2 points for not having some sort of filter, or something that at least attempts to be a filter

I suspect that some students will define their own version of stream-enumerate-interval – this is fine, but you should mark down if there are mistakes in that definition.

1. Explain why we use streams instead of lists in this procedure. (It is not an issue with correctness.) Hint: Consider the input 1000000000.

If we used lists, we would construct the entire list of numbers from 2 to n, filter the entire list, and then choose the first element. This is very inefficient in some cases, where the smallest factor is found early on but just never returned, such as 1000000000. The stream version does not suffer from this deficiency.

Just looking for the idea – that this could be very inefficient with lists.

0 points for saying that you can’t create infinite lists – the question specifically says in part a that students should not use infinite streams.