1. Write down exactly what will be printed by the following code executed by an applicative order Scheme interpreter.

(call-with-current-continuation
 (lambda (c)
   (map (lambda (x)
         (begin
           (display "fun ")
           (display x)
           (newline)
           (if (< x 0) (c x))
         x)
   )
   '(54 0 37 -9 245 -19) ) )

2. We wish to enhance the imperative flavor of FLK! by adding a looping construct (apart from new, read and write):

   \[
   E ::= \ldots | (\text{loop } N \ E) | (\text{loop } E) | (\text{break } N \ E) | (\text{break } E)
   \]

   $N$ is an optional label given to a loop. $(\text{loop } N \ E)$ and $(\text{loop } E)$ evaluate $E$ repeatedly forever. $(\text{break } E)$ ends the nearest lexically enclosing loop with the value of $E$. $(\text{break } N \ E)$ breaks the nearest lexically enclosing loop labeled $N$, returning with the value of $E$.

   (a) Write an iterative factorial function using these constructs together with new, read and write.

   (b) Give standard (denotational) semantics (i.e., $E$ with a continuation parameter) for these new loop/break constructs. (You can ignore the “store” parameter required for the new, read and write constructs, and you need not write $E$ for the store-accessing constructs.)