Write all code in Scheme/Scam unless otherwise directed. Assignment and explicit returns are not allowed.

These questions are worth one point each.

1. A functional programming language is characterized by:
   (A) the ability to define classes
   (B) the lack of assignment (and other side effects)
   (C) the lack of static typing
   (D) the ability to define functions

2. In Scheme (not Scam), the names of built-in functions and special forms can be redefined.
   (A) true and true
   (B) false and true
   (C) true and false
   (D) false and false

3. Scheme operators have precedence (e.g. multiplications are performed before additions) and associativity (e.g. arguments to variadic operators are preferentially combined from one direction).
   (A) true and true
   (B) false and false
   (C) true and false
   (D) false and true

4. One hallmark of an iterative process is that it:
   (A) cannot be implemented in functional languages
   (B) uses a while loop or a for loop
   (C) executes in constant space
   (D) does not exhibit recursion

5. What kind of process does the following function implement?
   ```scheme
   (define (f n k)
     (cond
       ((< n 2) (+ k n))
       ((= (% n 2) 0) (+ 1 (f (/ n 2) (* 2 k))))
       (else (f (- n 1) (+ k 1)))
     )
   )
   ```
   (A) both tail and non-tail recursive, but overall recursive
   (B) only non-tail recursive, so recursive
   (C) both tail and non-tail recursive, but overall iterative
   (D) only tail recursive, so iterative

6. What kind of process does the following function implement?
   ```scheme
   (define (f n)
     (f (+ n 1))
   )
   ```
   (A) iterative
   (B) recursive
   (C) neither, since it falls into an infinite recursive loop
7. What kind of process does the following function implement?

\[
\begin{align*}
\text{(define (f a t)} \\
\text{  (cond)} \\
\text{    ((= a 0) t)} \\
\text{    (else)} \\
\text{      (f (- a 1) (+ t (f (/ a 2) 1)))} \\
\text{  )}) \\
\end{align*}
\]

(A) both tail and non-tail recursive, but overall iterative
(B) only tail recursive, so iterative
(C) only non-tail recursive, so recursive
(D) both tail and non-tail recursive, but overall recursive

This next set of problems are worth 11 points each.

8. Suppose \( g \) is defined as:

\[
\begin{align*}
\text{(define (g f)} \\
\text{  (cond)} \\
\text{    ((and (integer? f) (= f 0)) 1)} \\
\text{    ((integer? f) (+ f (g (- f 1))}) \\
\text{    (else (f 3))} \\
\text{  )}) \\
\end{align*}
\]

To what does the expression \( (g g) \) evaluate and did the call spawn off a recursive or iterative process?

9. Define a function named \textit{product} that multiplies the integers between a given lower bound (inclusive) and a given upper bound (inclusive). For example, the expression:

\[
\text{(product 1 5)}
\]

should evaluate to 120. Your implementation must implement an iterative process.

10. Define a function named \textit{function+} that “adds” two functions together and returns this composition. For example:

\[
\text{(function+ cube double 3)}
\]

should evaluate to 216, assuming reasonable implementations of the functions \textit{cube} and \textit{double}. Note that \textit{function+} returns a function and that 3 was doubled and then that result was cubed.

This next set of problems are worth 20 points each.

11. Section 1.3.2 in the text discusses how to use lambdas in place of locally defined variables. Rewrite this function, in the same style as the text, using a lambda to replace the local definitions of \( a \) and \textit{square}.

\[
\begin{align*}
\text{(define (f x)} \\
\text{  (define a (+ x 1))} \\
\text{  (define (square x) (* x x))} \\
\text{  (square (* a a))} \\
\text{  )}) \\
\end{align*}
\]

Your new definition must be semantically equivalent to the old definition. You will need to rewrite the definition of \textit{square} to use a lambda, so that both \( a \) and \textit{square} have the same form.
12. Consider the higher-order function named `fractionSum`:

```scheme
(define (fractionSum f g count)
  (if (= count 0)
      0
      (+ (/ (f count) (g count)) (fractionSum f g (- count 1)))
  )
)
```

and the equation \( \ln 2 = \frac{1}{1} - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - \ldots \). Give a call to `fractionSum` such that it evaluates to an approximation of \( \ln 2 \) using 10 terms. You must use lambdas as the first two arguments. Note: passing a count of 1 should result in a value of \( \frac{1}{1} \). Note also: `fractionSum` uses integer division.

13. Define a function named \( s \) that uses recursion, testing for eveness, addition, and halving. Make use of the following facts:

\[
\begin{align*}
  s(a, b, t) &= t \text{ if } b \text{ is } 0 \\
  s(a, b, t) &= s(a + a, b/2, t) \text{ if } b \text{ is even} \\
  s(a, b, t) &= s(a + a, b/2, t + a) \text{ otherwise (Note: integer division)}
\end{align*}
\]

Define the functions \( s \), `even?`, and `halve`, all of which should do their arithmetic with addition and subtraction. Your `even?` and `halve` functions should take \( O(n) \) time. Your functions need only work for non-negative integers and your helper functions need not be nested.