Programming Languages
Exam 1

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

These questions are worth one point each.

1. T or F: The more expressive a language is, the more powerful it is (in a Church-Turing sense).
2. Which of the following is not a feature of the lambda calculus:
   (A) there is no assignment operation
   (B) functions can have only one formal parameter
   (C) looping is limited to recursion
   (D) the literals in the calculus are limited to integers
3. Consider both the names of built-in functions and special forms. In Scheme (not Scam), one can define variables initialized to the values bound to these names. Examples: (define v +) or (define v cond).
   (A) false and true, respectively
   (B) false and false, respectively
   (C) true and true, respectively
   (D) true and false, respectively
4. Scheme operators have associativity (e.g., arguments to variadic operators are preferentially combined from one direction) and precedence (e.g., multiplications are performed before additions).
   (A) true and false, respectively
   (B) true and true, respectively
   (C) false and true, respectively
   (D) false and false, respectively
5. One hallmark of a non-parallel, recursive process is that it:
   (A) executes in constant space
   (B) executes in space proportional to the maximum depth of recursion
   (C) executes in space proportional to its time complexity
   (D) executes in linear space
6. What kind of process does the following function implement?

   (define (f n k)
     (cond
      ((< n 2) (+ k n))
      ((= (% n 2) 0) (f (- n 1) (* 2 k)))
      (else (f (- n 1) (+ k 1)))
     )
   )

   (A) syntactically tail and non-tail recursive, but overall recursive
   (B) syntactically tail and non-tail recursive, but overall iterative
   (C) syntactically non-tail recursive, so recursive
   (D) syntactically tail recursive, so iterative
7. What kind of process does the following function implement?

```scheme
(define (f n)
  (f (+ n 1))
)
```

Choose the most precise answer.

(A) recursive  
(B) iterative  
(C) neither, since it falls into an infinite recursive loop  
(D) iterative, because the number of calls is limited by the recursion depth

8. What kind of process does the following function implement?

```scheme
(define (f a t)
  (cond
    ((= a 0) t)
    (else
      (f (- a 1) (+ t (f (/ a 2) t))))
  )
)
```

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

This next set of problems are worth 10 points each.

9. Convert this function to a semantically equivalent one that has the same time complexity but executes in constant space:

```scheme
(define (sum a b)
  (cond
    ((> a b) 0)
    (else (+ b (sum a (- b 1))))
  )
)
```

You must follow the original logic as closely as possible (e.g., b counts down and adds to the growing sum). Hint: determine how many base cases and how many formals that change.

10. Define a function named `factHat2` that multiplies the squares of the integers between one and a given upper bound (inclusive). For example, the expression:

```scheme
(factHat2 3)
```

should evaluate to 36 ($1^2 \times 2^2 \times 3^2$). Your implementation must implement an iterative process.

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 `square`.

```scheme
(define (f x)
  (define a (+ x 1))
  (define (square x) (* x x))
  (square (* a a))
)
```

Your new definition must be a direct syntactic transformation (e.g., the body of the function beyond the locals cannot change). You will need to syntactically transform the definition of `square` to use a lambda, so that both `a` and `square` have the same form.

This next set of problems are worth 20 points each.
12. Define a function named \texttt{function+} that “adds” two two-argument functions and returns this composition. In addition to the two-argument functions, \texttt{function+} is passed one of the arguments to each of the functions, as well. The value returned by \texttt{function+} is a new function that takes a single argument. For example:

\[
\texttt{((function+ pow 2 * 3) 4)}
\]

should evaluate to 144, because 4 times 3 is 12 and 12 raised to the second power is 144. Your implementation should preserve the order implied by the example.

13. Consider the higher-order function named \texttt{cfrac}, which computes the value of a continued fraction:

\[
\texttt{(define (cfrac f g count)}
\]

\[
\texttt{ (define (iter store src)}
\]

\[
\texttt{ (cond}
\]

\[
\texttt{ ((= src 0) store)}
\]

\[
\texttt{ (else (iter (/ (g src) (+ (f src) store)) (- src 1)))}
\]

\[
\texttt{ )}
\]

\[
\texttt{ (iter 0.0 count)}
\]

\[
\texttt{ )}
\]

and the equation:

\[
\pi = \frac{4}{1 + \frac{1^2}{2 + \frac{3^2}{2 + \frac{5^2}{2 + \frac{7^2}{2 + \ldots}}}}}
\]

Give a call to \texttt{cfrac} such that it evaluates to an approximation of \(\pi\) with an initial count of \(z\). You must use lambdas as the first two arguments. Note: passing a count of 0 should result in a value of 0. Passing a count of 1 should result in a value of \(0 + \frac{4}{1+\frac{1}{2}}\). Passing a count of 2 should result in a value of \(0 + \frac{4}{1+\frac{1}{2+\frac{3}{2}}}\). Please note that \texttt{cfrac} uses integer division, but to compute \(\pi\) properly, your functions \(f\) and \(g\) will need to force real number division. Hint: the function bound to \(f\) will have one special case, 1, while the function bound to \(g\) will have one special case, 4. The other cases for these functions follow a pattern.

There will be, at least, a 5 point deduction per for off-by-one errors.

14. (8 points) Define a function named \(g\), whose only the basic mathematical operations are allowed and whose formal parameters are \(x, y, r, s, \) and \(t\). If \(t\) is zero, then the value of \(r\) is returned. If \(t\) is even, \(g\) is called recursively. In this recursive call, \(r\) and \(s\) remain unchanged while \(x\) becomes \(x^2 + y^2\), \(y\) becomes \(2xy + y^2\), and \(t\) halves. If \(t\) is odd, then \(g\) is also called recursively. In this recursive call, \(x\) and \(y\) remain unchanged while \(r\) becomes \(sy + ry + rx\), \(s\) becomes \(sx + ry\), and \(t\) decrements.

(4 points) Give tight upper and lower bounds for the \texttt{space} complexity of \(g\) in the worst case. Assume constant space for holding the values of the formal parameters.

(4 points) Give tight upper and lower bounds for the \texttt{time} complexity of \(g\) in the worst case. Assume constant time for performing the mathematical operations at each iteration.

(4 points) Assuming \(r\) and \(s\) start out at zero and one, respectively, and \(x\) and \(y\) start out at zero and one, respectively, what are the first five return values of \(g\) (i.e., \(t = 0...4\))?