1. Using pure lambda expressions without recursion or \( Y \), design a function which takes as input a positive number \( n \) in the form of a Church numeral and outputs the smallest positive number \( k \) such that \( k! \) (the factorial of \( k \)) exceeds \( n \) (i.e., \( k! > n \)). Try to find a short and simple solution.
2. In a Scheme-like programming language, the following code

\[
\text{(let ((x 2))}
  \text{(cons (set! x (1+ x))}
  \text{(list (set! x (1+ x))))})
\]

evaluates to the list (3 2). Write down standard semantics (using continuations and store) for the construct \((\text{cons } E_1 E_2)\) in this language. Assume that lists are represented in memory by cells holding pairs, where the left item in the pair is a list element and the right item is the location of the next pair.

3. Exactly what will the following Java code print when executed? Explain why in up to three lines.

```java
public class Stack {
    static void rec(int x) throws Exception {
        if ( x > 5 ) throw new Exception("exc=" + x);
        rec(x+1);
        System.err.println("out=" + x);
    }
    public static final void main(String args[]) {
        try { rec(1); }
        catch ( Exception pex ) { System.err.println(pex); }
    }
}
```
4. Suppose you have to write programs in a preliminary release of Java which is missing the `finally` construct. In Java, if you write

```java
try { Et } finally { Ef }
```

you want `Ef` to execute no matter what, after `Et` completes either normally or abnormally with an exception. Give the simplest (least typing for the programmer) workaround recipe to achieve the effect of `finally` using any other Java features.

5. What will be the result of interpreting the following expressions in Scheme? Give up to three lines of explanations for each case.

(a) `(let ((r 1) (top (call-with-current-continuation (lambda (c) c))))
     (top (lambda (x) x)) r ) )`

(b) `(let ((r 1) (top (call-with-current-continuation (lambda (c) c))))
     (top top) (begin (display r) (newline) (set! r (1+ r)) ) ) )`
(c) Make minimal modifications to the code above so that it prints integers between 1 and 100 and then terminates. Complete the code shown below and write it out in full separately (do not fill the blanks in place). Some blanks may need to remain empty.

```
(let (____
     (top (call-with-current-continuation (lambda (c) c))))
  (____
   (____) (begin (display r) (newline)
                            (set! r (1+ r))) ) ) )
```

6. Infer the most specific type of all variables and subexpressions of the following expression:

```
(proc (f g x y) (if (f 3 y) (f x "static") (g x)))
```
7. In class we saw how to reconstruct \( \mathcal{R} \) the type of the expression \( \text{proc} \ I \ E \).

(a) Extend \( \mathcal{R} \) so that it can reconstruct types for expressions of the form \( \text{rec} \ I \ E_p \).

(b) Draw an abstract syntax tree for the following expression and mark type variables on appropriate nodes. Also write down all constraints between type expressions induced by the syntax tree. Pick as few fresh type variables as possible.

\[
(\text{rec fib} \ (\text{proc} \ n) \\
\quad (\text{if} \ (< 2 \ n) \ 1 \\
\quad (+ (\text{call fib} \ (- n 1)) \ (\text{call fib} \ (- n 2))))))
\]
8. In some language with array support, the following code

```plaintext
let
    f = proc(x, y) { y = y+1; x = 10; };
    a = array [ 200, 201, 202 ]; // first index is 0
    i = 0;
begin
    f(a[i], i);    print a;
    // f(i, a[i]); print i; print a;
```

prints

`[ 200, 10, 202 ]`

(a) Does the language have applicative or normal order evaluation? Justify your answer in two lines.

(b) What would be printed if the last line were to be uncommented?