1. Using the substitution model, show the evaluation of the following Scheme expression in detail. You should show all the evaluation steps, except that you may evaluate all the subexpressions of a combination which are variable lookups or primitives in one step, e.g. (+ 5 3 8) ⇒ 

\[
\begin{bmatrix}
\text{plus}
\end{bmatrix}
\begin{bmatrix}
5
3
8
\end{bmatrix}
\]

\[
(((\text{lambda} (n) (\text{if} (> n 3) + *)) 8) 4 2)
\]

\[\leftarrow\]
2. Consider the following function $t$:

\[
\begin{align*}
\text{(define } t &= \text{(lambda } ((f \ <\text{function}>)) \\
& \hspace{1cm} (\text{lambda } (x) \\
& \hspace{2cm} (f \ (f \ (f \ x)))))
\end{align*}
\]

For each of the following expressions, give the value that results from evaluating the expression, and justify your answer. You need \textit{not} do the whole evaluation formally using the substitution model, but should give a (brief) explanation why your answer is correct.

There are three parts to this problem.

(a) \((t \ \text{add1}) \ 0\) \hspace{2cm} \Leftarrow

(b) \((t \ (t \ \text{add1})) \ 0\) \hspace{2cm} \Leftarrow

(c) \(((t \ t) \ \text{add1}) \ 0\) \hspace{2cm} \Leftarrow
3. The transpose of a matrix $M$ is the matrix $M^T$ obtained by flipping $M$ about its main diagonal, so that its rows become columns, and its columns become rows. Formally, this means that if $a_{i,j}$ denotes the element of $M$ in row $i$ and column $j$, then the transpose takes $a_{i,j}$ to $a_{j,i}$.

For instance, the transpose of the matrix:

$$M = \begin{pmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
10 & 11 & 12
\end{pmatrix}$$

is the matrix:

$$M^T = \begin{pmatrix}
1 & 4 & 7 & 10 \\
2 & 5 & 8 & 11 \\
3 & 6 & 9 & 12
\end{pmatrix}$$

One way to represent a matrix in Scheme is as a list of rows, in which each row is a list of elements. In this representation, the first matrix above would be represented:

$$((1 \ 2 \ 3) \ (4 \ 5 \ 6) \ (7 \ 8 \ 9) \ (10 \ 11 \ 12))$$

and its transpose would be:

$$((1 \ 4 \ 7 \ 10) \ (2 \ 5 \ 8 \ 11) \ (3 \ 6 \ 9 \ 12))$$

Note that the first list of the transpose consists of the first element of each list in the original matrix; the second list of the transpose consists of the second element of each list in the original matrix, etc.

Write a Scheme function `transpose` which takes a matrix, represented as a list of lists, and returns its transpose represented as a list of lists. You may assume that all of the lists in the input matrix have the same number of elements. Feel free to use built-in Scheme functions such as `append`, `filter`, `map`, `foldr`, `reverse`, `member`, etc. if they are useful.
4. **Algorithmic Analysis** Suppose you are given a function \( \text{insert} \ x \ L \) which, given an object \( x \) and a proper list \( L \), returns a new list consisting of the elements of \( L \) with \( x \) inserted into the correct location according to some ordering relation. Suppose \( \text{insert} \) runs in \( O(n) \) time in the worst case, where \( n \) is the length of \( L \). Given that this is so, what is the worst-case asymptotic runtime of the following implementation of the \( \text{sort} \) function?

Justify your answer.

\[
\begin{align*}
\text{(define sort} \\
\quad \text{(lambda ((L <lst>))} \\
\quad \quad \text{(if (null? L)} \\
\quad \quad \quad \text{'}()) \\
\quad \quad \text{(insert (car L)} \\
\quad \quad \quad \text{(sort (cdr L)))}} \text{))}
\end{align*}
\]
5. **Short Answer**

(a) Draw a box-and-pointer diagram to illustrate the data structure that would be created by evaluation of the following expression:

\[
\text{define } L \text{ (cons (cons (cons 1 2) (cons 3 4)) '()))}
\]

(b) Is the structure pointed to by \(L\) in the problem above a *proper* list? Why or why not?
(c) For a class example, we implemented rational numbers. To do this, we created a new type `<rat>` as follows:

```
(define-class <rat> ()
  (numer <integer>)
  (denom <integer>))
```

Why is it preferable to define the `<rat>` type in this way, as opposed to simply writing
```
(define <rat> <pair>), then defining numer and denom as aliases for car and cdr?  
```

(d) Why doesn’t evaluating the following Scheme expression generate a “division by zero” error?

```
(lambda () (/ 1 0))
```

Unless otherwise specified, please format all code examples as `Code` blocks.
6. For each of the following expressions, provide a definition of blob that would cause the expression to evaluate to 23. For instance, if the expression were:

(car blob)

...then one possible answer would be to write:

(define blob (cons 23 0))

There is more than one right answer in each case—please provide only one solution for each expression.

(a) (let* ((grog blob) (blob grog)) (+ blob grog 1))

(b) (foldr + 0 (map (lambda (n) (- n 1)) blob))

(c) (cadr (filter blob '(5 18 21 23 37)))

(d) ((blob blob) blob)

(e) (+ 3 (* 4 (length (cdr blob)))))
A graph $G = (V, E)$ is a mathematical structure consisting of a set $V$ of vertices and a set $E$ of edges connecting those vertices. For instance, here is a graph with four vertices and eight edges:

![Graph Diagram]

In Problem Set 3, we represented graphs by explicitly keeping a list of vertices, and a separate list of edges (edges, you recall, were represented by structures of the form $\langle u, v \rangle$, with $u$ being the vertex where the edge begins and $v$ being the vertex where the edge ends).

A different way to represent a graph is using a data structure known as an adjacency list. An adjacency list (as the name suggests) is a list, which has one element for each vertex in the graph. The element of the adjacency list corresponding to a particular vertex $u$ is itself a list, containing the names of all the vertices $u$ is adjacent to in the graph. Vertex $u$ is adjacent to vertex $v$ if there is an edge from $u$ to $v$.

For instance, the entry for vertex $a$ in the above example graph would be the list $(b, c, d)$, since those are the three vertices to which edges go starting from $a$.

Suppose you have been given a `<graph>` class implemented using an adjacency list. Each vertex is represented as a `<symbol>`. You are given an accessor `(neighbors vtx G)` which, given a graph $G$ and a vertex name $vtx$ returns a list of the vertices adjacent to $vtx$. Furthermore, you are given an accessor `(vertices G)` which returns a list of the vertices of graph $G$.

Using these abstractions, write a function `(inbound vtx G)` which, given a vertex name $vtx$ and a graph $G$, returns a list (possibly empty) of all the vertices from which there is an edge leading to $vtx$ in the graph.

You should feel free to use built-in Scheme functions such as `append`, `filter`, `map`, `foldr`, `reverse`, `member`, etc. if they are useful to you.
8. Write a function \((k\text{-subsets } L \ k)\), which takes a proper list \(L\) and an integer \(k \geq 0\). It should return a list of all the lists of \(k\) elements that can be formed from the objects in \(L\). For example,

\[
(k\text{-subsets } '(a b c d) 2)
\]

...should return:

\[
((a b) (a c) (a d) (b c) (b d) (c d))
\]

Note that your function does not have to return the \(k\)-lists in the same order as is given here. \(\Leftarrow\)