This handout contains sample problems for the first exam, which will be held in class on Thursday, 3 October 2002.

Don’t panic – there are more problems in this sample exam than there will be on the actual exam.

Solutions will be distributed in class on Tuesday, 1 October 2002.
Problem 1  What will Scheme print in response to the following statements? Assume that they are each evaluated in order in a single Scheme buffer. Write your answer below each statement. You may write “procedure” if a procedure would be returned, or “error” if an error message would be returned. (This problem spans two pages.)

```
(define x 2)

x

(x)

(define (y) (* x 2))

y

(y)
```
(define (a) (lambda (x) (+ x 1)))

(a)

(let ((x 1)
   (y 2)
   (z (+ x 4)))
   (+ x y z))

(define (if a b c) (+ a b c))

(if 2 3 4)
Problem 2  Write a function called new-add which returns the sum of two numbers. Do not use the internal functions + and -, but instead, use inc (an already defined Scheme procedure which takes one argument and returns the sum of that argument and 1) and dec (a similar procedure which returns the sum of its argument and −1).

Is your procedure recursive?  

Is your procedure tail-recursive?  

Is the process it generates recursive or iterative?
Problem 3  Assume that we have defined sum and square as follows:

\[
\text{(define (sum term a next b)} \\
\text{  (if (> a b)} \\
\text{    0} \\
\text{    (+ (term a) \\
\text{      (sum term (next a) next b)))))}
\]

\[
\text{(define (square x) (* x x))}
\]

Determine the order of growth in time for the following functions using \( \Theta \) notation. (Hint: you will only need to use one or more of the following for your answers: \( \Theta(1) \), \( \Theta(\log n) \), \( \Theta(n) \), \( \Theta(n^2) \) and \( \Theta(2^n) \). All classes might not be used.) Write your answer in the blanks to the right of each function.

\[
\text{(define (integrate abfd x )} \\
\text{  (sum (lambda (x) (* (f x) dx)) a \\
\text{   (lambda (x) (+ x dx)) b))}
\]

\[
\text{(define (number-of-bits-in n)} \\
\text{  (if (< n 2)} \\
\text{    1} \\
\text{    (+ 1 (number-of-bits-in (/ n 2)))))}
\]

\[
\text{(define (times-5 x) } \\
\text{  (* x 5))}
\]

\[
\text{(define (exp a b)} \\
\text{  (cond ((< b 0) (error "Oops! b cannot be negative")))) \\
\text{   (1)} \\
\text{   (else (if (odd? b) \\
\text{     (* a (exp a (- b 1))) \\
\text{     (square (exp a (/ b 2)))))))})
\]

\[
\text{(define (sum-of-squares x y)}} \\
\text{  (+ (square x) \\
\text{   (square y))))}
\]

\[
\text{(define (triangle-sum n)}} \\
\text{  (sum (lambda (m) (sum (lambda (x) x) 1 inc m)) 1 inc n)}
\]
Problem 4 Write a procedure `power-close-to` that takes two non-zero positive integers (b and n) as arguments and returns the smallest power of b that is greater than n. That is, it should return the smallest integer i such that \( b^i > n \). You may use the Scheme procedure `(expt b i)` which raises b to the power i.

Does your procedure generate an iterative process or a recursive process?
Problem 5  A local bookstore has contracted aD University to provide an inventory system for their web
site. We can create a database of books using Scheme. The constructor for a single book will be called
make-book and takes the name of a book and its price as parameters.

    (define (make-book name price)
        (cons name price))

Write the selectors book-name and book-price.

The inventory of books will be stored in a list. The selectors for our inventory data structure are first-book
and rest-books, defined as follows:

    (define first-book car)
    (define rest-books cdr)

Write the constructor make-inventory.

Draw the box-and-pointer diagram that results from the evaluation of

    (define store-inventory
        (make-inventory (make-book 'sicp 60)
            (make-book 'collecting-pez 15)
            (make-book 'the-little-schemer 35)))
Problem 5 (continued) Write a procedure called `find-book` which takes the name of a book and an inventory as parameters and returns the book’s data structure (name and price) if the book is in the store’s inventory, and nil otherwise.

The bookstore has asked us to change our system to include a count of the number of copies of each book the store has on hand. We redefine our book constructor as follows:

```
(define (make-book name price num-in-stock)
  (list name price num-in-stock))
```


Will `find-book` need to be changed to accommodate our new representation?
Problem 5 (continued)  Now that we are storing the number of copies in stock, write a procedure called in-stock that takes a book name and an inventory as the parameters, and returns #t if at least one copy of the book is in stock, or #f otherwise. If the book is not listed in the inventory at all, in-stock should also return #f. You may want to use your find-book procedure from above.
Problem 6  Write a function \texttt{add-n} of one argument \texttt{n} that returns a procedure. The returned procedure takes one argument \texttt{x} and returns the sum of \texttt{x} and \texttt{n}.

Using \texttt{add-n}, and without using the built-in Scheme procedure \texttt{*}, write \texttt{mult} which takes two integer arguments \texttt{a} and \texttt{b} and returns their product.
Problem 7  Assume the following expressions have been evaluated in the order they appear.

```
(define a (list (list 'q) 'r 's))
(define b (list (list 'q) 'r 's))
(define c a)
(define d (cons 'p a))
(define e (list 'p (list 'q) 'r 's))
```

Complete the table below with the result of applying the functions `eq?`, `eqv?`, and `equal?` to the two expressions on the left of each row. For example, the elements of the top row will represent the result from evaluating `(eq? a c)`, `(eqv? a c)`, and `(equal? a c)`. Your result should be written as #t, #f or `undefined`.

<table>
<thead>
<tr>
<th><code>&lt;operand_1&gt;</code></th>
<th><code>&lt;operand_2&gt;</code></th>
<th><code>eq?</code></th>
<th><code>eqv?</code></th>
<th><code>equal?</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td>(cdr d)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>e</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(car a)</td>
<td>(car e)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(car a)</td>
<td>(cadr e)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(caar a)</td>
<td>(caadr e)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem 8  Write occurrences, a procedure of two arguments s and tree that returns the number of times the first argument (an atom) appears in the second (a tree). You may find accumulate-tree, shown below, to be helpful.

```
(define (accumulate-tree tree term combiner null-value)
  (cond ((null? tree) null-value)
        ((not (pair? tree)) (term tree))
        (else (combiner (accumulate-tree (car tree) term combiner null-value)
                       (accumulate-tree (cdr tree) term combiner null-value))))
```

Quiz 1 Fall 2002
Problem 9  Draw the box and pointer diagrams for the following structures.

'(1)

'(1 2 3 4)

'(((1)))

'((1 . 2) (3 . 4))

'(1 (2 3) (4 (5) (6 7))))