This is a 50 minute exam, commencing at 10:10am and finishing at 11:00am. You may not use any reference material during the exam, including books or notes. You may not use any electronics equipment (cell phones, calculators, computers, etc.) during the exam. *Please read every question carefully before answering* and express your answers as completely as you can.

There are 4 questions on the exam, each worth 10 points; **you may choose** *any three* **to complete**. Thus the highest score that can be achieved on the exam is 30 points. Please indicate which 3 questions you wish to be graded in the check boxes next to the question numbers below.

Question	Grade?	Score
1		
2		
3		
4		
Total		

- 1. (10 points.) Define SCHEME functions with the following specifications.
 - (a) (2 points.) Define a Scheme function $\max 2$ which takes two numeric inputs (call them x and y) and returns the larger of the two. (You may not use the built-in scheme function \max for this purpose—define your own function from scratch using a conditional.)

(b) (2 points.) Define a SCHEME function max3 which takes three numeric inputs (call them x, y, and z) and returns the largest of the three. (You may not use the built-in scheme function max for this purpose, but you may use your function max2 from the previous problem.)

(c) (2 points.) Define a SCHEME function crazy which takes a single input x and returns

$$\frac{(x+10)(x+10)+10}{x+10}.$$

(d) (2 points.) Define a SCHEME function monster-fact which takes a single numeric argument x and returns (x!)!. (Yes, that's the factorial of the factorial of x, so (monster-fact 4) should return (4!)! = 24! = 620448401733239439360000. You must define the factorial function from scratch, if you intend to use it.)

(e) (2 points.) Define a Scheme function dfact (which stands for "double factorial"). The double factorial function is defined (for the natural numbers $\{0, 1, 2, ...\}$) by the recursive rule:

$$dfact(n) = \begin{cases} 1 & \text{if } n = 0, \\ 1 & \text{if } n = 1, \\ n \cdot dfact(n-2) & \text{otherwise.} \end{cases}$$

2. (**10 points.**) In this problem, you will implement *Halley's method* for extracting square roots. Consider the following mysterious function of two variables *Q* and *g*:

$$h(Q,g) = \frac{g(g^2 + 3Q)}{3g^2 + Q}.$$

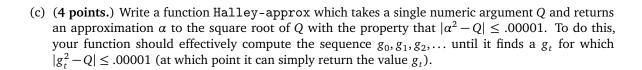
Edmond Halley (after whom the comet is named) noticed the following remarkable fact. For any (positive) number Q and any (positive) number g, h(Q,g) is always closer to \sqrt{Q} than g was. In particular, for any specific fixed value of Q we can consider the sequence of numbers

$$g_s = \begin{cases} 1 & \text{if } s = 0, \\ h(Q, g_{s-1}) & \text{for } s > 0. \end{cases}$$

Then these numbers g_0, g_1, \ldots converge very quickly to \sqrt{Q} .

(a) (1 **points.**) Write a Scheme function h which computes the function above (so, the function h should take two arguments *Q* and *g*).

(b) (3 points.) Write a function Halley-iterate which takes two numeric arguments, Q and t, and returns g_t (defined by the sequence above). Thus, if t=0, your function should return 1; if t=1, your function should return $g_1=h(Q,1)$, etc.



(d) (**2 points.**) Show how to restructure your code from the previous problems so that *h* is defined in the scope of Halley-approx, and use this restructuring to make *h* a function of a single parameter *g*.

3. (10 points.) The integers 1, 2, 4, and 5 can be written as the sum of two perfect squares:

$$1 = 0^2 + 1^2$$
, $2 = 1^2 + 1^2$, $4 = 0^2 + 2^2$, and $5 = 1^2 + 2^2$.

On the other hand, neither 3, 6, nor 7 can be expressed this way.

In this problem, you will define a function sum-of-squares so that (sum-of-squares n) returns #t if the positive integer n can be written as a sum of squares of two integers and #f otherwise. You may assume that n is positive. If you wish, you may use the following function is-square, which returns #t if k is a perfect square, and #f otherwise:

(a) (2 points.) Define a SCHEME function square-pieces so that (square-pieces x n) returns #t if both x and (n-x) are perfect squares, and #f otherwise.

(b) (4 points.) Observe that n can be written as a sum of two squares exactly when there is a number $x \in \{0, ..., n\}$ for which (square-pieces x n) returns #t. Write a SCHEME function (test-upto k n) which returns #t if there is a number $x \in \{0, ..., k\}$ for which (square-pieces x n) is true.

(c) (4 points.) Using the above functions, define the SCHEME function sum-of-squares. For full credit, indicate how the definitions of your helper functions can be made private; do you need to keep passing around the parameter n?

4. (10 points.)

(a) (2 points.) Consider the following version of factorial:

```
(define (factorial n)
  (let ((recursive-value (factorial (- n 1))))
    (if (= n 0)
         1
         (* n recursive-value))))
```

For which, if any, values of n does this correctly compute n!? Explain.

(b) (2 points.) Consider the following declaration:

```
(define (f x)
  (define (g y) (+ x y))
  (define (h x) (+ x (g x)))
  (h (+ x 10))
```

After this, what would (f 100) return?

(c) (2 points.) To what does the following expression evaluate?

```
(let ((x 10)
	(y 20)
	(z 40))
	(let ((x (+ x 10))
		(y (+ x 20)))
	(+ z (- x y))))
```

(d) (2 points.) Consider the following two implementations of multiplication (the first is called times; the second is called ftimes).

```
(define (times x y)

(cond ((= x 0) 0)

((= y 0) 0)

(else (+ x (times x (- y 1))))))
```

and

These both correctly compute multiplication (if the arguments are non-negative integers). However, the behavior of a call to (times 1000000 1000001) will be quite different from that of a call to (ftimes 1000000 1000001). Explain.

(e) (2 points.) Consider the following flawed implementation of the Fibonacci sequence.

For which values, if any, of *n* will this correctly compute the *n*th Fibonacci number? What's wrong?

SCRATCH SPACE

SCRATCH SPACE