Round 1

Lambda Calculus

500

It is the value of the following expression:

((lambda (+ -) (* + -))
 (* 3 6)
((lambda (/ ^) (* ^ /)) 4 6)
(l lambda (- *) (+ - *))))

What is 42?

Orders of Growth

400

The orders of growth in time and space of:

(define (h n)
 (if (= n 0)
 1
 (+ (h (quotient n 3))
  (h (quotient n 3)))))

What are $\Theta(n)$ in time and $\Theta(\log n)$ in space?

500

This is the minimum order of growth of any comparison-based sorting algorithm.

What is $\Theta(n \log n)$

Higher-Order Procedures

200

It is the type of the following procedure:

(define (test foo bar n)
 (if (bar n)
  (+ 1 (foo n))
  (test foo bar (+ n 3)))))

What is (number -> number), (number -> boolean), number -> number?
300
   It is the type of the following procedure:
   (define (test a b)
   (lambda (x) (a (b x))))

   What is (A -> B), (C -> A) -> (C -> B)?

400
   If double is a procedure that takes a procedure of one argument and returns a
   procedure that applies the original procedure twice, this is the value returned by:

   (((double (double double)) inc) 5)

   What is 21?

500
   It is the value of the following expression:
   (define (foo x k)
   (if (= x 0) (k 0)
   (foo (- x 1) (lambda (y)
   (k (+ y x))))))
   (foo 5 (lambda (y) y))

   What is 15?

Lists

300
   It is the value of the following expression:

   (define x '(a b x))
   (define y (list x x (list 'x x)))
   (set-cdr! (cdr y) (list (quote x)))
   y

   What is ((a b x) (a b x) x)?
If we were to implement cons, car, and cdr as procedures, by writing cons as a procedure of its two arguments:
(define (cons x y)
  (lambda (m) (m x y)))

this is how cdr would be defined.

What is
(define (cdr l)
  (l (lambda (x y) y)))

It is the value of the following expression:

(let ((foo 'a))
  `(list foo ,foo ,'foo ',foo))

What is (list foo a foo (quote a))?

Round 2
Data

The problem with the following fragment of code:

(define make-vector cons)
(define vector-x car)
(define vector-y cdr)
(define v1 (make-vector 2 3))
(define (magnitude v)
  (let ((cars (* (car vec) (car vec)))
         (cdrs (* (cdr vec) (cdr vec)))
         (sqrt (+ cars cdrs))))

What is an abstraction violation?
It is the length of the list foo:

```
(define foo (list 1))
(set-car! foo 2)
(set-cdr! foo `(3 ,4 ,foo))
```

What is infinity?

The value of the following stream:

```
(define foo
  (cons-stream 1
    (add-streams foo foo)))
```

What is the powers of two?

The mathematical definition of the \textit{i}th element of this stream:

```
(define foo
  (cons-stream 1
    (add-streams foo (stream-cdr ints)))))
```

What is the sum of integers from 1 to \textit{i}?

**Environment Model**

In a lexically scoped language like Scheme, this is, by definition, where free variables in procedures passed as arguments are looked up:

* in the environment where the procedure is called.
* in the environment where the lambda expression was evaluated.
* in the global environment.
* in the primitive list in the global environment.
* in Billings, Montana.

What is "in the environment where the lambda expression was evaluated?"
During procedure applications, if we always extend the environment in which the combination was evaluated, rather than extend the environment pointed to by the double-bubble, then we are using this kind of scoping rule.

What is dynamic?

Object-Oriented Programming

In the following code, dairy-product inherits from this other class:

(define (make-dairy-product self name temp)
  (let ((container 'none)
         (bad #f)
         (scent 'lemon)
         (food-part (make-food self name temp)))
    (make-handler 'dairy-product
      (make-methods
        'NAME (lambda () name)
        'SCENT (lambda () scent)
        'SPOILED? (lambda () (set! scent 'vile) #t)
        food-part)))

What is food?

We require all constructors (e.g. make-foo) to accept an argument named self, in order to maintain a pointer from each handler to this.

What is the instance?

It is the value of the final expression below:

(define (make-kid self)
  (let ((root-part (make-root-object self)))
    (make-handler 'kid)
(make-methods
 'MALE? (lambda () (not (ask self 'female?)))
 'FEMALE? (lambda () (not (ask self 'male?)))
 root-part))

(ask (create-kid) 'female?)

What is an error (out of stack space)?

Meta-Circular Evaluator
400
It is the value of the following expression in a dynamic-binding Scheme:

(let ((x 20))
 (let ((f (lambda (y) (- y x))))
  (let ((x 10))
   (f 30))))

What is 20?

600
The number of times m-eval is invoked when the following expression is entered into the evaluator:

(((lambda (x) (* x 2)) 3)

What is 7 (combination, lambda, 3, (* x 2), *, x, 2)?

800
The three functions to modify in the evaluator to handle define statements of the form:

(<variable> := <binding>)

What are definition?, definition-variable, and definition-value (optionally make-define)?