MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001 Structure and Interpretation of Computer Programs
Spring, 2007

Practice Problems, March 1

Types Dr. Kimberle Koile

For each expression or set of expressions, give the value and type of the value returned by evaluating the last expression in the set.

	value	type
1. ((lambda (x) (+ x y)) 7)	error, undefined variable y	
2. ((lambda (x) (let ((y 4)) (+ x y))) 7)	11	number
3. (lambda (x) (x 4 5))	compound procedure	number, number → A → A
4. (lambda (a b c) (+ a b))	compound procedure	number, number, any → number
5. (lambda (x y) (lambda (x) (y x)))	compound procedure	A, A→B → A→B
6. (((lambda (x y) (lambda (z) (x y z))) + 2) 4)	6	number

Using the substitution model, first substitute + and 2 for x and y, and call the outer level lambda: ((lambda (z) (+ 2 z)) 4); then call the lambda with 4 bound to z to get (+ 2 4). Note that without the 4 and the outer parens, the value is a compound procedure: (lambda (z) (+ 2 z)), with type num->num,.

```
                                          value                          type

7.  ((lambda (x)
        (let ((a  1)
              (b  5))
      (if  x  a  b))
      )
     (> 20 10))                           1                              number
```
The lambda is called with the value of (> 20 10), with is #t; so the value of a is returned.

```
8.  (define  x  +)
    (let ((a  3))
      (list  x  a  a))                    (+  3  3)               pair(num->num, list(num))
```
The return value is not evaluated; the list is just constructed. The type is a pair rather than a list because the list type is specified with one type for its members. So the type of (+ 3 3) also would be list(A), but the above pair type is more specific, so preferred.

```
9.   (define (foo a  b)
        (let ((x  6)
              (c (+  a  5)))
           (+  b  x  c)))

     ((lambda (x  y  f)
         (f  x  y))
       1  2  foo)                          14                            number
```

The lambda is called with arguments that bind x to 1, y to 2, f to the procedure foo.
The expression (foo 1 2) is then evaluated to get 14.

Extra problem (not to worry about now):

```
10.  (let ((a  10)
           (b  2))
       (let ((c  (+ a  b)))
         (*  a  c)))                      120                           number
```

Note: The second let is needed because the value of a variable is not bound until the entire list of variable-value pairs is evaluated. In this example, the value of a or b can't be used in defining c in the first let's list of variables.