

Distribution of Scores

21-25:	24	
26-30:	29 30	
31-35:	32 34 34	
36-40:	38 38 40	Average: 38.6
41-45:	41 42 43 45	Median: 40
46-50:	46 46 47 47	



Problem 1 (18 points)

Pick *three* of the following terms and for each term: (i) define it *and* (ii) give an example. (If you do more than three, all will be graded, *but* only the *lowest* three scores will count.)

- (a) bound variable
- (b) contours
- (c) expressed value
- (d) shadowed variables
- (e) thunk

Answers:

- (a) bound variable, see EOPL page 10.
- (b) contours, see EOPL page 89.
- (c) expressed value, see EOPL page 61.
- (d) shadowed variables, see EOPL page 89.
- (e) thunk, see EOPL page 136.

Problem 2 (8 points) Consider the following code.

```
let x = 10
    y = 1
in let f = proc(x) +(x,y)
    g = proc(t,y) +(t,y)
    y = 2
in
    (f (g x 3))
```

What is the value of this expression under

- (i) *lexical scoping*?
- (ii) *dynamic scoping*?

Answer:

- (i) *lexical scoping*: 13 14 — when evaluating the body of *f*, the *y* refers the *y* declared on line 2.
- (ii) *dynamic scoping*: 13 15 — when evaluating the body of *f*, the *y* refers the *y* declared on line 5.



Problem 3 (12 points) Consider the following code.

```
let a = 1    b = 10
in let p = proc(x,y,z) begin set x = +(a,x); set y = +(y,10);
    set z = +(z,10); print(x,y) end
in begin (p a b b); print(a,b) end
```

Fill in the table below with the values printed under each of the listed parameter passing schemes.

call-by-	print(x,y)	print(a,b)
(a) value		
(b) reference		
(c) value-result		

call-by-	print(x,y)	print(a,b)
(a) value	2 20	1 10
(b) reference	2 30	2 30
(c) value-result	2 20	2 20

In the call-by-reference case, in the execution of *p*'s body: *x* and *a* are aliased and *y*, *z*, and *b* are aliased. In the call-by-value-result case, in the execution of *p*'s body: things go along as in the call-by-value case to the very end at which point the value of *x* is copied back to *a* and (the tricky part) the values of *y* and *z* (both of which have value 20) are copied back to *b*.

Problem 4 (12 points) Consider the following code.

```

let count = 0   j = 1   k = 10
  in let add1 = proc(x) begin set count = +(count,1);   +(x,1) end
      add2 = proc(x) begin set count = +(count,10);  +(x,2) end
      add3 = proc(x) begin set count = +(count,100); +(x,3) end
      in let p = proc(r,s,t,u)
          begin set r = (add1 r); set s = +(t,t); print(s,t); end
          in begin
              (p j k (add2 j) (add3 j)); print(j,count)
            end

```

Fill in the table below with the values printed under each of the listed parameter passing schemes.

call-by-	print(s,t)	print(j,count)
(a) value		
(b) name		
(c) need		

call-by-	print(s,t)	print(j,count)
(a) value	6 3	1 111
(b) name	8 4	2 31
(c) need	8 4	2 11

In all cases, count keeps track of how many times each `add n` is executed (+1 for each `add1`; +10 for each `add2`; and +100 for each `add3`). In the call-by-value case, it is once each for each of the `add n` 's. In the call-by-name case, `add1` is evaluated once, `add2` is evaluated 3 times (for each time `t` is evaluated), and `add3` is never evaluated since we never evaluate `u`. In the call-by-name case in the execution of `p`'s body: `r` and `j` are aliased, `s` and `k` are aliased; `t` is equivalent to `(add2 j)`, a In the call-by-need case, it is the same story as the call-by-name case, except we only ever evaluate the expression named by `t` once.