Name: **PID:**

Quiz 3

Consider the following pseudo code. Suppose side-effects are allowed, and that operands of '+' are evaluated left-to-right.

```
GLOBAL int y = 1;

PROCEDURE inc(int x)
BEGIN
  WRITE('incrementing ' + x);
  x += y;
  return x;
END

PROCEDURE print_if_positive(int x, int y)
BEGIN
  IF (x > 0) THEN
    BEGIN
      WRITE('x=' + x + ', y=' + y);
    END
END

MAIN PROGRAM
BEGIN
  int y =  10;
  int a = -10;
  int b =   1;

  print_if_positive(inc(a), inc(y));
  print_if_positive(inc(b), inc(y));

END
```

Please answer the questions on the reverse side. You can use this side for scratch space; anything on this side will not be graded.

Provide the output produced by the program on the reverse side assuming:

A) call-by-value, lexical scoping, and eager left-to-right evaluation. [2 points]

```
incrementing -10    (value of a copied into inc)
incrementing 10     (value of y-MAIN copied into inc)
incrementing 1      (value of b copied int inc)
incrementing 10     (value of y-MAIN copied into inc)
x=2, y=11           (inc(a) returned 2, so the result is printed)
                    (incremented by y-GLOBAL, so just one)
```

B) call-by-value, dynamic scoping, and eager left-to-right evaluation. [2 points]

```
incrementing -10    (value of a copied into inc)
incrementing 10     (value of y-MAIN copied into inc; y-MAIN is not changed (c.-b.-value)
incrementing 1      (value of b copied into inc)
incrementing 10     (value of y-MAIN copied into inc)
x=11, y=20          (incremented by y-MAIN, so +10)
```

C) call-by-reference, lexical scoping, and eager left-to-right evaluation. [2 points]

```
incrementing -10    (a referenced from inc -> new value is -9)
incrementing 10     (y-MAIN referenced from inc -> new value is 11)
incrementing 1      (b referenced from inc -> new value is 2)
incrementing 11     (y-MAIN referenced from inc -> new value is 12)
x=2, y=12
```

D) dynamic scoping and normal-order evaluation (i.e., call-by-name). [2 points]

```
incrementing -10    (inc(a) is evaluated for 'x > 0' condition -> false: inc(y) is not eval'd)
incrementing 1      (inc(b) is evaluated for 'x > 0' condition -> true, new value is 11
incrementing 11     (inc(b) is evaluated for + operator -> new value is 21
incrementing 10     (inc(y) is evaluated for + operator -> new value is 20)
x=21, y=20
```

This assumes that dynamic name resolution avoids infinite recursion by skipping over the 'y' parameter to 'print_if_positive'. Stating that an infinite results also gave full credit.

E) lexical scoping and lazy evaluation. [2 points]

```
incrementing -10    (inc(a) is evaluated for 'x > 0' condition -> false: inc(y) is not eval'd)
incrementing 1      (inc(b) is evaluated for 'x >0' condition -> true)
incrementing 10     (inc(y) is evaluated for + operator, y-GLOBAL used for increment)
x=2, y=11
```

inc(b) is not evaluated twice because the result is "cached" and reused.

Please stop by during office hours if you would like to see an in-depth explanation.