**Exercise 1.5.** Ben Bitdiddle has invented a test to determine whether the interpreter he is faced with is using *applicative-order evaluation* or *normal-order evaluation*. He defines the following two procedures:

```
(define (p) (p))

(define (test x y)
  (if (= x 0)
      0
      y))
```

Then he evaluates the expression

```
(test 0 (p))
```

What behavior will Ben observe with an interpreter that uses *applicative-order evaluation*? What behavior will he observe with an interpreter that uses *normal-order evaluation*? Explain your answer. (Assume that the evaluation rule for the special form `if` is the same whether the interpreter is using *normal* or *applicative* order: The predicate expression is evaluated first, and the result determines whether to evaluate the consequent or the alternative expression.)

**Solution.** Well, this is interesting! In brief, Ben will find that the interpreter never terminates with no output while using the *applicative-order evaluation*. In *normal-order evaluation*, he will see that interpreter prints an `0` immediately after the procedures being executed. The reason is presented as the following:

According to the description of *applicative-order evaluation* and *normal-order evaluation* in section 1.1.5, the *applicative-order evaluation* model would not applied the operator to the operands until both have been evaluated. On the contrary, the *normal-order evaluation* methord indicates that the interpreter would not evaluate the operands until their value were needed. In other words, so long as the value of some particular operands is not needed at present, the *normal-order evaluation* methord will continue to substitute operand expression without noticing them.

As shown in Figure 1, the *applicative-order evaluation* strategy keeps the interpreter evaluating the operand `p`, whose value is unavailable here. Because, under the environment of *applicative-order evaluation* model, the expression `(test 0 (p))` here is identical to itself after being extracted. Undoutfully, this puts the interpreter in keeping evaluating the expression `(test 0 (p))` infinitely in Environment **I** . Thus, nothing is output:

```
(test 0 (p))
(test 0 (p))
...
(test 0 (p))
...
```

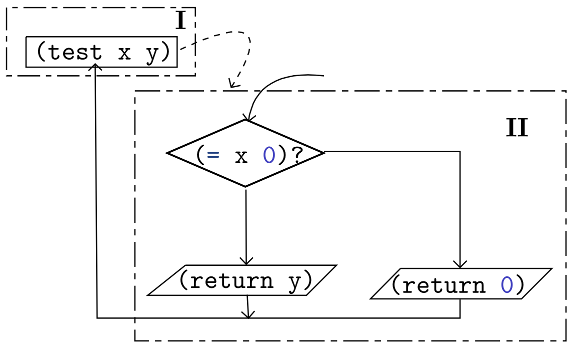

**Figure 1.** Flow Chart for Exercise 1.5

By comparision, in the *normal-order evaluation* model, `p` in expression `(test 0 (p))` here would not be evaluated until it is needed.

Obviously, as is the case in Figure 1, the interpreter will proceed to Environment **II** immediately and reveal its procedure as the following:

```
(if (= 0 0) 0 p)
(if #t 0 p)
(0)
```

Consequently, `0` will be output immediately after the procedures above being executed.