```
(define (special-form? x)
  (or (equal? x 'quote)
      (equal? x 'lambda)
      (equal? x 'let)
      (equal? x 'if)))

(define (get-first-binding-side bindings side)
  (cond ((equal? side 'left) (car (car bindings)))
        ((equal? side 'right) (car (cdr (car bindings))))))

(define (get-first-binding-replacement bindings)
  (car (cdr (cdr (car bindings)))))

(define (remove-binding-val val bindings side)
  (cond ((null? val) bindings)
        ((null? bindings) bindings)
        ((equal? (get-first-binding-side bindings side) val) (remove-binding-val val (cdr bindings) side))
        (else (cons (car bindings) (remove-binding-val val (cdr bindings) side)))))

(define (remove-lambda-bindings params bindings side)
  (cond ((null? params) bindings)
        ((null? bindings) bindings)
        (else (remove-lambda-bindings (cdr params) (remove-binding-val (car params) bindings side) side))))

(define (remove-let-bindings params bindings side)
  (cond ((null? params) bindings)
        ((null? bindings) bindings)
        (else (remove-let-bindings (cdr params) (remove-binding-val (car (car params)) bindings side) side))))

(define (replace-bindings-let-params params bindings side)
  (cond ((null? params) params)
        ((null? bindings) params)
        (else (cons (cons (car (car params)) (replace-bindings (cdr (car params)) bindings side))
                    (replace-bindings-let-params (cdr params) bindings side)))))

(define (replace-bindings val bindings side)
  (cond ((null? bindings) val)
        ((null? val) val)
        ((pair? val)
         (cond ((equal? (car val) 'quote) val)
               ((equal? (car val) 'lambda) (list
                                            (car val)
                                            (car (cdr val))
                                            (replace-bindings (car (cdr (cdr val))) (remove-lambda-bindings (car (cdr val)) bindings side) side)))
               ((equal? (car val) 'let) (list
                                         (car val)
                                         (replace-bindings-let-params (car (cdr val)) bindings side)
                                         (replace-bindings (car (cdr (cdr val))) (remove-let-bindings (car (cdr val)) bindings side) side)))
               (else (cons (replace-bindings (car val) bindings side) (replace-bindings (cdr val) bindings side)))))
        ((equal? (get-first-binding-side bindings side) val) (get-first-binding-replacement bindings))
        (else (replace-bindings val (cdr bindings) side))))

(define (nullify-binding-val val bindings side)
  (cond ((null? bindings) bindings)
        ((null? val) bindings)
        ((equal? (get-first-binding-side bindings side) val)
         (cond ((equal? side 'left) (cons (list '() (car (cdr (car bindings))) (car (cdr (cdr (car bindings)))) (nullify-binding-val val (cdr bindings) side)))
               ((equal? side 'right) (cons (list (car (car bindings)) '() (car (cdr (cdr (car bindings)))) (nullify-binding-val val (cdr bindings) side)))))
        (else (cons (car bindings) (nullify-binding-val val (cdr bindings) side)))))

(define (nullify-binding-vals leftvals rightvals bindings)
  (cond ((and (null? leftvals) (null? rightvals)) bindings)
        (else (nullify-binding-vals (cdr leftvals) (cdr rightvals) (nullify-binding-val (car leftvals) (nullify-binding-val (car rightvals) bindings 'right) 'left)))))

(define (concat-symbols x y)
  (string->symbol (string-append (symbol->string x) "!" (symbol->string y))))

(define (new-lambda-bindings-helper leftparams rightparams bindings)
  (cond ((and (null? leftparams) (null? rightparams)) bindings)
        ((equal? (car leftparams) (car rightparams)) (new-lambda-bindings-helper (cdr leftparams) (cdr rightparams) bindings))
        (else (new-lambda-bindings-helper (cdr leftparams) (cdr rightparams) (cons (list (car leftparams) (car rightparams) (concat-symbols (car leftparams) (car rightparams))) bindings)))))

(define (new-lambda-bindings leftparams rightparams bindings)
  (new-lambda-bindings-helper leftparams rightparams (nullify-binding-vals leftparams rightparams bindings)))

(define (let-bindings-flatten params flattened)
  (cond ((null? params) flattened)
        (else (let-bindings-flatten (cdr params) (cons (car (car params)) flattened)))))

(define (new-let-bindings leftparams rightparams bindings)
  (new-lambda-bindings (let-bindings-flatten leftparams '()) (let-bindings-flatten rightparams '()) bindings))

(define (replace-let-bindings params new-bindings old-bindings side)
  (cond ((null? params) params)
        (else (cons (cons (replace-bindings (car (car params)) new-bindings side)
                          (replace-bindings (cdr (car params)) old-bindings side))
                    (replace-let-bindings (cdr params) new-bindings old-bindings side)))))

(define (expr-compare-bindings x y bindings)
  (cond ((and (pair? x) (pair? y))
         (cond ((not (equal? (length x) (length y))) (list 'if '% (replace-bindings x bindings 'left) (replace-bindings y bindings 'right)))
               ((and (special-form? (car x)) (special-form? (car y)))
                (cond ((not (equal? (car x) (car y))) (list 'if '% (replace-bindings x bindings 'left) (replace-bindings y bindings 'right)))
                      ((and (equal? (car x) 'quote) (equal? (car y) 'quote))
                       (cond ((equal? (cdr x) (cdr y)) x)
                             (else (list 'if '% x y))))
                      ((and (equal? (car x) 'lambda) (equal? (car y) 'lambda))
                       (cond ((not (equal? (length (car (cdr x))) (length (car (cdr y))))) (list 'if '% (replace-bindings x bindings 'left) (replace-bindings y bindings 'right)))
                             (else (list
                                    (car x)
                                    (expr-compare-bindings
                                     (replace-bindings (car (cdr x)) (new-lambda-bindings (car (cdr x)) (car (cdr y)) bindings) 'left)
                                     (replace-bindings (car (cdr y)) (new-lambda-bindings (car (cdr x)) (car (cdr y)) bindings) 'right)
                                     '())
                                    (expr-compare-bindings (car (cdr (cdr x))) (car (cdr (cdr y))) (new-lambda-bindings (car (cdr x)) (car (cdr y)) bindings))))))
                      ((and (equal? (car x) 'let) (equal? (car y) 'let))
                       (cond ((not (equal? (length (car (cdr x))) (length (car (cdr y))))) (list 'if '% (replace-bindings x bindings 'left) (replace-bindings y bindings 'right)))
                             (else (list
                                    (car x)
                                    (expr-compare-bindings
                                     (replace-let-bindings (car (cdr x)) (new-let-bindings (car (cdr x)) (car (cdr y)) bindings) bindings 'left)
                                     (replace-let-bindings (car (cdr y)) (new-let-bindings (car (cdr x)) (car (cdr y)) bindings) bindings 'right)
                                     '())
                                    (expr-compare-bindings (car (cdr (cdr x))) (car (cdr (cdr y))) (new-let-bindings (car (cdr x)) (car (cdr y)) bindings))))))
                      (else (cons (expr-compare-bindings (car x) (car y) bindings) (expr-compare-bindings (cdr x) (cdr y) bindings)))))
               ((or (special-form? (car x)) (special-form? (car y))) (list 'if '% (replace-bindings x bindings 'left) (replace-bindings y bindings 'right)))
               (else (cons (expr-compare-bindings (car x) (car y) bindings) (expr-compare-bindings (cdr x) (cdr y) bindings)))))
        ((or (pair? x) (pair? y)) (list 'if '% (replace-bindings x bindings 'left) (replace-bindings y bindings 'right)))
        ((equal? (replace-bindings x bindings 'left) (replace-bindings y bindings 'right)) (replace-bindings x bindings 'left))
```

```
        ((and (equal? x #t) (equal? y #f)) '%)
        ((and (equal? x #f) (equal? y #t)) '(not %))
        (else (list 'if '% (replace-bindings x bindings 'left) (replace-bindings y bindings 'right)))))

(define (expr-compare x y)
  (expr-compare-bindings x y '()))

(define (test-expr-compare x y)
  (and (equal? (eval x) (eval `(let ((% #t)) ,(expr-compare x y))))
       (equal? (eval y) (eval `(let ((% #f)) ,(expr-compare x y))))))

(define test-expr-x
  '(list #t #t #f #f 3 4
         (list "a" "b")
         (list 'a)
         '(a b)
         (if 0 5 4)
         (let ((a 3) (b 2)) (+ a b))
         (((lambda (f)
             ((lambda (x)
                (x x))
              (lambda (h)
                (f (lambda (a) ((h h) a))))))
           (lambda (f)
             (lambda (n)
               (if (= n 0) 1 (* n (f (- n 1)))))))) 10)))

(define test-expr-y
  '(list #t #f #t #f 4 4
         (list "a" "b" 'c)
         (list 'a)
         '(a c)
         (+ 0 5 4)
         (let ((a 3) (c 4)) (/ c a))
         (((lambda (f)
             ((lambda (z)
                (z z))
              (lambda (t)
                (f (lambda (arg) ((t t) arg))))))
           (lambda (f)
             (lambda (n)
               (if (< n 2) 1 (+ (f (- n 1)) (f (- n 2)))))))) 4)))
```