Scheme

● Graded

Group

Sangwon Ji
Seong Jae Ahn
View or edit group

Total Points

32 / 29 pts

Autograder Score

32.0 / 29.0

Autograder Results

```
=====================================================================
Assignment: Project 4: Scheme Interpreter
OK, version v1.18.1
=====================================================================

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Scoring tests

---------------------------------------------------------------------
Understanding Eval/Apply
Passed: 0
Failed: 0
[k.........] 0.0% passed

---------------------------------------------------------------------
Problem 1
Passed: 2
Failed: 0
[ooooooooook] 100.0% passed

---------------------------------------------------------------------
Problem 2
Passed: 1
Failed: 0
[ooooooooook] 100.0% passed

---------------------------------------------------------------------
Problem 3
Passed: 2
Failed: 0
[ooooooooook] 100.0% passed

---------------------------------------------------------------------
Problem 4
Passed: 1
```

```
Failed: 0
[ooooooooook] 100.0% passed

----------------------------------------------------------------------
Problem 5
Passed: 3
Failed: 0
[oooooooooook] 100.0% passed

----------------------------------------------------------------------
Problem 6
Passed: 2
Failed: 0
[oooooooooook] 100.0% passed

----------------------------------------------------------------------
Problem 7
Passed: 2
Failed: 0
[oooooooooook] 100.0% passed

----------------------------------------------------------------------
Problem 8
Passed: 2
Failed: 0
[oooooooooook] 100.0% passed

----------------------------------------------------------------------
Problem 9
Passed: 3
Failed: 0
[oooooooooook] 100.0% passed

----------------------------------------------------------------------
Problem 10
Passed: 2
Failed: 0
[oooooooooook] 100.0% passed

----------------------------------------------------------------------
Problem 11
Passed: 2
Failed: 0
[oooooooooook] 100.0% passed

----------------------------------------------------------------------
Problem 12
Passed: 2
Failed: 0
[oooooooooook] 100.0% passed

----------------------------------------------------------------------
Problem 13
Passed: 2
```

```
Failed: 0
[ooooooooook] 100.0% passed

---------------------------------------------------------------------
Problem 14
Passed: 3
Failed: 0
[ooooooooook] 100.0% passed

---------------------------------------------------------------------
Scheme tests in tests.scm

Score: 1.0/1

---------------------------------------------------------------------
Problem 15
Passed: 1
Failed: 0
[ooooooooook] 100.0% passed

---------------------------------------------------------------------
Problem 16
Passed: 1
Failed: 0
[ooooooooook] 100.0% passed

---------------------------------------------------------------------
Problem EC 1
Passed: 1
Failed: 0
[ooooooooook] 100.0% passed

---------------------------------------------------------------------
Point breakdown
Understanding Eval/Apply: 0.0/0
Problem 1: 1.0/1
Problem 2: 2.0/2
Problem 3: 2.0/2
Problem 4: 2.0/2
Problem 5: 1.0/1
Problem 6: 1.0/1
Problem 7: 2.0/2
Problem 8: 2.0/2
Problem 9: 2.0/2
Problem 10: 1.0/1
Problem 11: 2.0/2
Problem 12: 2.0/2
Problem 13: 2.0/2
Problem 14: 2.0/2
tests.scm: 1.0/1
Problem 15: 2.0/2
Problem 16: 2.0/2
Problem EC 1: 2.0/2
```

Score:
Total: 31.0

Cannot backup when running ok with --local.

Final Score:32.0
Early Submission. Bonus Point Included

Submitted Files

questions.scm

Download

```
(define (caar x) (car (car x)))
(define (cadr x) (car (cdr x)))
(define (cdar x) (cdr (car x)))
(define (cddr x) (cdr (cdr x)))

;; Problem 15
;; Returns a list of two-element lists
(define (enumerate s)
  ; BEGIN PROBLEM 15
   (define (helper lst index)
    (if (null? lst) '()
        (cons (list index (car lst))
              (helper (cdr lst) (+ index 1))))
    ) (helper s 0)
)

  ; END PROBLEM 15

;; Problem 16

;; Merge two lists S1 and S2 according to ORDERED? and return
;; the merged lists.
(define (merge ordered? s1 s2)
  ; BEGIN PROBLEM 16
  (if (null? s2)
      s1
      (if (null? s1)
          s2
       (if (ordered? (car s2) (car s1))
       (cons(car s2)(merge ordered? (cdr s2) s1))
      (cons (car s1) (merge ordered? s2 (cdr s1)))))))

  ; END PROBLEM 16

;; Optional Problem

;; Returns a function that checks if an expression is the special form FORM
(define (check-special form)
  (lambda (expr) (equal? form (car expr))))

(define lambda? (check-special 'lambda))
(define define? (check-special 'define))
(define quoted? (check-special 'quote))
(define let?    (check-special 'let))

;; Converts all let special forms in EXPR into equivalent forms using lambda
(define (let-to-lambda expr)
  (cond ((atom? expr)
         ; BEGIN OPTIONAL PROBLEM
```

```
        'replace-this-line
        ; END OPTIONAL PROBLEM
        )
       ((quoted? expr)
        ; BEGIN OPTIONAL PROBLEM
        'replace-this-line
        ; END OPTIONAL PROBLEM
        )
       ((or (lambda? expr)
            (define? expr))
        (let ((form   (car expr))
              (params (cadr expr))
              (body   (cddr expr)))
          ; BEGIN OPTIONAL PROBLEM
          'replace-this-line
          ; END OPTIONAL PROBLEM
          ))
       ((let? expr)
        (let ((values (cadr expr))
              (body   (cddr expr)))
          ; BEGIN OPTIONAL PROBLEM
          'replace-this-line
          ; END OPTIONAL PROBLEM
          ))
       (else
        ; BEGIN OPTIONAL PROBLEM
        'replace-this-line
        ; END OPTIONAL PROBLEM
        )))

; Some utility functions that you may find useful to implement for let-to-lambda

(define (zip pairs)
  'replace-this-line)

```

scheme_classes.py Download

```
import builtins

from pair import *


class SchemeError(Exception):
    """Exception indicating an error in a Scheme program."""

##################
# Environments #
##################


class Frame:
    """An environment frame binds Scheme symbols to Scheme values."""

    def __init__(self, parent):
        """An empty frame with parent frame PARENT (which may be None)."""
        self.bindings = {}
        self.parent = parent

    def __repr__(self):
        if self.parent is None:
            return '<Global Frame>'
        s = sorted(['{0}: {1}'.format(k, v) for k, v in self.bindings.items()])
        return '<{{{0}}} -> {1}>'.format(', '.join(s), repr(self.parent))

    def define(self, symbol, value):
        """Define Scheme SYMBOL to have VALUE."""
        # BEGIN PROBLEM 1
        self.bindings[symbol] = value
        # END PROBLEM 1

    def lookup(self, symbol):
        """Return the value bound to SYMBOL. Errors if SYMBOL is not found."""
        # BEGIN PROBLEM 1
        if symbol in self.bindings: # check from define
            return self.bindings[symbol]
        elif self.parent:
            return self.parent.lookup(symbol)
        else:
        # END PROBLEM 1
            raise SchemeError('unknown identifier: {0}'.format(symbol))

    def make_child_frame(self, formals, vals):
        """Return a new local frame whose parent is SELF, in which the symbols
        in a Scheme list of formal parameters FORMALS are bound to the Scheme
        values in the Scheme list VALS. Both FORMALS and VALS are represented
        as Pairs. Raise an error if too many or too few vals are given.
```

```
50
51          >>> env = create_global_frame()
52          >>> formals, expressions = read_line('(a b c)'), read_line('(1 2 3)')
53          >>> env.make_child_frame(formals, expressions)
54          <{a: 1, b: 2, c: 3} -> <Global Frame>>
55          """
56          if len(formals) != len(vals):
57              raise SchemeError('Incorrect number of arguments to function call')
58          # BEGIN PROBLEM 8
59          frame = Frame(self)
60
61          while formals is not nil:
62              frame.define(formals.first, vals.first)
63              formals, vals = formals.rest , vals.rest
64          return frame
65          # END PROBLEM 8
66
67  ###############
68  # Procedures #
69  ###############
70
71
72  class Procedure:
73      """The the base class for all Procedure classes."""
74
75
76  class BuiltinProcedure(Procedure):
77      """A Scheme procedure defined as a Python function."""
78
79      def __init__(self, py_func, need_env=False, name='builtin'):
80          self.name = name
81          self.py_func = py_func
82          self.need_env = need_env
83
84      def __str__(self):
85          return '#[{0}]'.format(self.name)
86
87
88  class LambdaProcedure(Procedure):
89      """A procedure defined by a lambda expression or a define form."""
90
91      def __init__(self, formals, body, env):
92          """A procedure with formal parameter list FORMALS (a Scheme list),
93          whose body is the Scheme list BODY, and whose parent environment
94          starts with Frame ENV."""
95          assert isinstance(env, Frame), "env must be of type Frame"
96
97          from scheme_utils import validate_type, scheme_listp
98          validate_type(formals, scheme_listp, 0, 'LambdaProcedure')
99          validate_type(body, scheme_listp, 1, 'LambdaProcedure')
100         self.formals = formals
101         self.body = body
```

```
102         self.env = env
103 
104     def __str__(self):
105         return str(Pair('lambda', Pair(self.formals, self.body)))
106 
107     def __repr__(self):
108         return 'LambdaProcedure({0}, {1}, {2})'.format(
109             repr(self.formals), repr(self.body), repr(self.env))
110 
111 
112 class MuProcedure(Procedure):
113     """A procedure defined by a mu expression, which has dynamic scope.
114      _________________
115     < Scheme is cool! >
116      -----------------
117             \   ^__^
118              \  (oo)\_______
119                 (__)\       )\/\
120                     ||----w |
121                     ||     ||
122     """
123 
124     def __init__(self, formals, body):
125         """A procedure with formal parameter list FORMALS (a Scheme list) and
126         Scheme list BODY as its definition."""
127         self.formals = formals
128         self.body = body
129 
130     def __str__(self):
131         return str(Pair('mu', Pair(self.formals, self.body)))
132 
133     def __repr__(self):
134         return 'MuProcedure({0}, {1})'.format(
135             repr(self.formals), repr(self.body))
136 
```

scheme_eval_apply.py Download

```
import sys
from pair import *
from scheme_utils import *
from ucb import main, trace

import scheme_forms

##############
# Eval/Apply #
##############


def scheme_eval(expr, env, _=None):  # Optional third argument is ignored
    """Evaluate Scheme expression EXPR in Frame ENV.

    >>> expr = read_line('(+ 2 2)')
    >>> expr
    Pair('+', Pair(2, Pair(2, nil)))
    >>> scheme_eval(expr, create_global_frame())
    4
    """
    # Evaluate atoms
    if scheme_symbolp(expr):
        return env.lookup(expr)
    elif self_evaluating(expr):
        return expr

    # All non-atomic expressions are lists (combinations)
    if not scheme_listp(expr):
        raise SchemeError('malformed list: {0}'.format(repl_str(expr)))
    first, rest = expr.first, expr.rest
    if scheme_symbolp(first) and first in scheme_forms.SPECIAL_FORMS:
        return scheme_forms.SPECIAL_FORMS[first](rest, env)

    else:
        # BEGIN PROBLEM 3
        "*** YOUR CODE HERE ***"
        operator3 = scheme_eval(first, env)
        operands3 = rest.map(lambda operand3: scheme_eval(operand3,env))
        result3 = scheme_apply(operator3,operands3,env)
        return result3
        # return scheme_apply(scheme_eval(first,env) , rest.map(lambda operand :
scheme_eval(operand, env)), env)

        # END PROBLEM 3


def scheme_apply(procedure, args, env):
    """Apply Scheme PROCEDURE to argument values ARGS (a Scheme list) in
```

```
    Frame ENV, the current environment."""
    validate_procedure(procedure)
    if not isinstance(env, Frame):
      assert False, "Not a Frame: {}".format(env)

    if isinstance(procedure, BuiltinProcedure):
        # BEGIN PROBLEM 2
        py_args = [] #new list
        while args is not nil:
            py_args.append(args.first)
            args = args.rest
        if procedure.need_env:
            py_args.append(env)
        # END PROBLEM 2
        try:
            # BEGIN PROBLEM 2
            return procedure.py_func(*py_args)

            # END PROBLEM 2
        except TypeError as err:
            raise SchemeError('incorrect number of arguments: {0}'.format(procedure))

    elif isinstance(procedure, LambdaProcedure):
        # BEGIN PROBLEM 9
        child_frame = procedure.env.make_child_frame(procedure.formals,args)
        return eval_all(procedure.body, child_frame)

        # END PROBLEM 9
    elif isinstance(procedure, MuProcedure):
        # BEGIN PROBLEM 11


        MU_frame = env.make_child_frame(procedure.formals, args)
        return scheme_eval(procedure.body, MU_frame)

        # END PROBLEM 11
    else:
        assert False, "Unexpected procedure: {}".format(procedure)


def eval_all(expressions, env):
    """Evaluate each expression in the Scheme list EXPRESSIONS in
    Frame ENV (the current environment) and return the value of the last.

    >>> eval_all(read_line("(1)"), create_global_frame())
    1
    >>> eval_all(read_line("(1 2)"), create_global_frame())
    2
    >>> x = eval_all(read_line("((print 1) 2)"), create_global_frame())
    1
    >>> x
    2
```

```
    >>> eval_all(read_line("((define x 2) x)"), create_global_frame())
    2
    """
    # BEGIN PROBLEM 6
    if expressions is nil:
        return None

    else:
        while expressions.rest != nil:
            scheme_eval(expressions.first, env)
            expressions = expressions.rest
        return scheme_eval(expressions.first, env)

    # END PROBLEM 6


##################
# Tail Recursion #
##################

class Unevaluated:
    """An expression and an environment in which it is to be evaluated."""

    def __init__(self, expr, env):
        """Expression EXPR to be evaluated in Frame ENV."""
        self.expr = expr
        self.env = env


def complete_apply(procedure, args, env):
    """Apply procedure to args in env; ensure the result is not an Unevaluated."""
    validate_procedure(procedure)
    val = scheme_apply(procedure, args, env)
    if isinstance(val, Unevaluated):
        return scheme_eval(val.expr, val.env)
    else:
        return val


def optimize_tail_calls(unoptimized_scheme_eval):
    """Return a properly tail recursive version of an eval function."""
    def optimized_eval(expr, env, tail = False):
        """Evaluate Scheme expression EXPR in Frame ENV. If TAIL,
        return an Unevaluated containing an expression for further evaluation.
        """

        # BEGIN PROBLEM EC
        sys.setrecursionlimit(1000000)  #increased limit larger than 501501

        while True:
            result = unoptimized_scheme_eval(expr, env)

```

```
153             if isinstance(result, Unevaluated):
154                 expr, env = result.expr, result.env
155             else:
156                 return result
157 
158     return optimized_eval
159 
160 #     # END PROBLEM EC
161 
162 
163 ######################################################################
164 # Uncomment the following line to apply tail call optimization #
165 ######################################################################
166 
167 scheme_eval = optimize_tail_calls(scheme_eval)
168 
```

scheme_forms.py Download

```
from scheme_eval_apply import *
from scheme_utils import *
from scheme_classes import *
from scheme_builtins import *

##################
# Special Forms #
##################

# Each of the following do_xxx_form functions takes the cdr of a special form as
# its first argument---a Scheme list representing a special form without the
# initial identifying symbol (if, lambda, quote, ...). Its second argument is
# the environment in which the form is to be evaluated.


def do_define_form(expressions, env):
    """Evaluate a define form.
    >>> env = create_global_frame()
    >>> do_define_form(read_line("(x 2)"), env) # evaluating (define x 2)
    'x'
    >>> scheme_eval("x", env)
    2
    >>> do_define_form(read_line("(x (+ 2 8))"), env) # evaluating (define x (+ 2 8))
    'x'
    >>> scheme_eval("x", env)
    10
    >>> # problem 10
    >>> env = create_global_frame()
    >>> do_define_form(read_line("((f x) (+ x 2))"), env) # evaluating (define (f x) (+ x 8))
    'f'
    >>> scheme_eval(read_line("(f 3)"), env)
    5
    """
    validate_form(expressions, 2)  # Checks that expressions is a list of length at least 2
    signature = expressions.first
    if scheme_symbolp(signature):
        # assigning a name to a value e.g. (define x (+ 1 2))
        validate_form(expressions, 2, 2)  # Checks that expressions is a list of length exactly 2
        # BEGIN PROBLEM 4
        variable_name = signature
        temp = scheme_eval(expressions.rest.first, env)
        env.define(variable_name, temp) #!!!
        return variable_name
        # END PROBLEM 4


    elif isinstance(signature, Pair) and scheme_symbolp(signature.first):
        # defining a named procedure e.g. (define (f x y) (+ x y))
        # BEGIN PROBLEM 10
```

```
50          func_name = signature.first
51          formals = signature.rest
52          formal_curr = formals
53
54          while formal_curr is not nil:
55              if not scheme_symbolp(formal_curr.first):
56
57                  raise SchemeError(f'invalid formal parameter: {formal_curr.first}')
58              formal_curr = formal_curr.rest
59
60          body = expressions.rest
61          lambda_proc = LambdaProcedure(formals, body, env)########AGAIN
62
63          env.define(func_name, lambda_proc)
64
65          return func_name
66
67          # END PROBLEM 10
68      else:
69          bad_signature = signature.first if isinstance(signature, Pair) else signature
70          raise SchemeError('non-symbol: {0}'.format(bad_signature))
71
72
73  def do_quote_form(expressions, env):
74      """Evaluate a quote form.
75
76      >>> env = create_global_frame()
77      >>> do_quote_form(read_line("((+ x 2))"), env) # evaluating (quote (+ x 2))
78      Pair('+', Pair('x', Pair(2, nil)))
79      """
80      validate_form(expressions, 1, 1)
81      # BEGIN PROBLEM 5
82      "*** YOUR CODE HERE ***"
83      return expressions.first
84      # END PROBLEM 5
85
86
87  def do_begin_form(expressions, env):
88      """Evaluate a begin form.
89
90      >>> env = create_global_frame()
91      >>> x = do_begin_form(read_line("((print 2) 3)"), env) # evaluating (begin (print 2) 3)
92      2
93      >>> x
94      3
95      """
96      validate_form(expressions, 1)
97      return eval_all(expressions, env)
98
99
100 def do_lambda_form(expressions, env):
101     """Evaluate a lambda form.
```

```
102
103     >>> env = create_global_frame()
104     >>> do_lambda_form(read_line("((x) (+ x 2))"), env) # evaluating (lambda (x) (+ x 2))
105     LambdaProcedure(Pair('x', nil), Pair(Pair('+', Pair('x', Pair(2, nil))), nil), <Global Frame>)
106     """
107     validate_form(expressions, 2)
108     formals = expressions.first
109     validate_formals(formals)
110     # BEGIN PROBLEM 7
111     body = expressions.rest
112     procedure = LambdaProcedure(formals, body, env)
113     return procedure
114
115
116     # END PROBLEM 7
117
118
119 def do_if_form(expressions, env):
120     """Evaluate an if form.
121
122     >>> env = create_global_frame()
123     >>> do_if_form(read_line("(#t (print 2) (print 3))"), env) # evaluating (if #t (print 2) (print 3))
124     2
125     >>> do_if_form(read_line("(#f (print 2) (print 3))"), env) # evaluating (if #f (print 2) (print 3))
126     3
127     """
128     validate_form(expressions, 2, 3)
129     if is_scheme_true(scheme_eval(expressions.first, env)):
130         return scheme_eval(expressions.rest.first, env)
131     elif len(expressions) == 3:
132         return scheme_eval(expressions.rest.rest.first, env)
133
134
135 def do_and_form(expressions, env):
136     """Evaluate a (short-circuited) and form.
137
138     >>> env = create_global_frame()
139     >>> do_and_form(read_line("(#f (print 1))"), env) # evaluating (and #f (print 1))
140     False
141     >>> # evaluating (and (print 1) (print 2) (print 4) 3 #f)
142     >>> do_and_form(read_line("((print 1) (print 2) (print 3) (print 4) 3 #f)"), env)
143     1
144     2
145     3
146     4
147     False
148     """
149
150     # BEGIN PROBLEM 12
151     if expressions is nil:
152         return True
153
```

```
    while expressions != nil:
        temp = scheme_eval(expressions.first,env)
        if temp is False:
            return False
        expressions = expressions.rest
    return temp
    # END PROBLEM 12


def do_or_form(expressions, env):
    """Evaluate a (short-circuited) or form.

    >>> env = create_global_frame()
    >>> do_or_form(read_line("(10 (print 1))"), env) # evaluating (or 10 (print 1))
    10
    >>> do_or_form(read_line("(#f 2 3 #t #f)"), env) # evaluating (or #f 2 3 #t #f)
    2
    >>> # evaluating (or (begin (print 1) #f) (begin (print 2) #f) 6 (begin (print 3) 7))
    >>> do_or_form(read_line("((begin (print 1) #f) (begin (print 2) #f) 6 (begin (print 3) 7))"), env)
    1
    2
    6
    """
    # BEGIN PROBLEM 12

    if expressions is nil:
        return False

    while expressions.rest != nil:
        temp = scheme_eval(expressions.first,env)
        if temp is not False:
            return temp

        expressions = expressions.rest

    return scheme_eval(expressions.first, env)
    # END PROBLEM 12


def do_cond_form(expressions, env):
    """Evaluate a cond form.

    >>> do_cond_form(read_line("((#f (print 2)) (#t 3))"), create_global_frame())
    3
    """
    while expressions is not nil:
        clause = expressions.first
        validate_form(clause, 1)
        if clause.first == 'else':
            test = True
            if expressions.rest != nil:
                raise SchemeError('else must be last')
```

```
206         else:
207             test = scheme_eval(clause.first, env)
208         if is_scheme_true(test):
209             # BEGIN PROBLEM 13
210 
211             if clause.rest is nil:
212               return test
213             temp = None
214 
215             current =clause.rest
216 
217             while current is not nil:
218                 expr = current.first
219                 temp = scheme_eval(expr, env)
220                 current = current.rest
221             return temp
222         expressions = expressions.rest
223 
224             # END PROBLEM 13
225 
226 
227 
228 def do_let_form(expressions, env):
229     """Evaluate a let form.
230 
231     >>> env = create_global_frame()
232     >>> do_let_form(read_line("(((x 2) (y 3)) (+ x y))"), env)
233     5
234     """
235     validate_form(expressions, 2)
236     let_env = make_let_frame(expressions.first, env)
237     return eval_all(expressions.rest, let_env)
238 
239 
240 def make_let_frame(bindings, env):
241     """Create a child frame of Frame ENV that contains the definitions given in
242     BINDINGS. The Scheme list BINDINGS must have the form of a proper bindings
243     list in a let expression: each item must be a list containing a symbol
244     and a Scheme expression."""
245     if not scheme_listp(bindings):
246         raise SchemeError('bad bindings list in let form')
247     names = vals = nil
248     # BEGIN PROBLEM 14
249 
250     check_name=set()
251 
252     while bindings is not nil:
253         binding = bindings.first
254 
255         if not scheme_listp(binding) or len(binding) != 2 or not scheme_symbolp(binding.first):
256             raise SchemeError("SchemeError binding: {}".format(str(binding)))
257 
```

```
258         if binding.first in check_name:
259             raise SchemeError("SchemeError binding: {}".format(str(binding.first)))
260 
261         check_name.add(binding.first)
262 
263         names = Pair (binding.first, names)
264         vals = Pair (scheme_eval(binding.rest.first, env), vals)
265 
266         bindings = bindings.rest
267 
268     # END PROBLEM 14
269     return env.make_child_frame(names, vals)
270 
271 
272 def do_quasiquote_form(expressions, env):
273     """Evaluate a quasiquote form with parameters EXPRESSIONS in
274     Frame ENV."""
275     def quasiquote_item(val, env, level):
276         """Evaluate Scheme expression VAL that is nested at depth LEVEL in
277         a quasiquote form in Frame ENV."""
278         if not scheme_pairp(val):
279             return val
280         if val.first == 'unquote':
281             level -= 1
282             if level == 0:
283                 expressions = val.rest
284                 validate_form(expressions, 1, 1)
285                 return scheme_eval(expressions.first, env)
286         elif val.first == 'quasiquote':
287             level += 1
288 
289         return val.map(lambda elem: quasiquote_item(elem, env, level))
290 
291     validate_form(expressions, 1, 1)
292     return quasiquote_item(expressions.first, env, 1)
293 
294 
295 def do_unquote(expressions, env):
296     raise SchemeError('unquote outside of quasiquote')
297 
298 
299 ##################
300 # Dynamic Scope #
301 ##################
302 
303 def do_mu_form(expressions, env):
304     """Evaluate a mu form."""
305     validate_form(expressions, 2)
306     formals = expressions.first
307     validate_formals(formals)
308     # BEGIN PROBLEM 11
309     return MuProcedure(formals, expressions.rest.first)
```

```
    # END PROBLEM 11


SPECIAL_FORMS = {
    'and': do_and_form,
    'begin': do_begin_form,
    'cond': do_cond_form,
    'define': do_define_form,
    'if': do_if_form,
    'lambda': do_lambda_form,
    'let': do_let_form,
    'or': do_or_form,
    'quote': do_quote_form,
    'quasiquote': do_quasiquote_form,
    'unquote': do_unquote,
    'mu': do_mu_form,
}

```