Com S 342 Name: ____________________
Spring 2005 TA (or Section): ____________________

Principles of Programming Languages

Exam 4 on Assignment, Parameters, Statements, and OO Features

This test has 5 questions and pages numbered 1 through 9.

Reminders

For this test, you can use one (1) page (8.5 by 11 inches, one (1) side, no less than 9pt font) of notes. Handwriting is okay. No photo-reduction is permitted. Don't use anything with printing on the other side, please. These notes are to be handed in at the end of the test. Have your name in the top right corner. Use of other notes or failure to follow these instructions will be considered cheating.

If you need more space, use the back of a page. Note when you do that on the front.

This test is timed. We will not grade your test if you try to take more than the time allowed. Therefore, before you begin, please take a moment to look over the entire test so that you can budget your time.

For programs, indentation is important to us for "clarity" points; if your code is sloppy or hard to read, you will lose points. Correct syntax also matters. Check your code over for syntax errors. You will lose points if your code has syntax errors.

You can use helping procedures whenever you like.

For Grading:

Problem	Points	Score
1	5	
2	10	
3	40	
4	20	
5	25	

1. (5 points) What is the difference between statements and expressions? Give a brief explanation.

2. Consider the following defined language expression.

```
letrec ohno(x) = (ohno x) % loop forever
in let test = proc(a,b,c) if a then a+b else 0
       true = 1
   in (test true 342 (ohno 227))
```

   (a) (5 points) Suppose in interpreter A, the above expression returns 343. Is the parameter passing mechanism in this interpreter lazy or eager? (Please write either "lazy" or "eager" for your answer.)

   (b) (5 points) Suppose in interpreter B, the above expression loops forever. Is the parameter passing mechanism in this interpreter lazy or eager? (Please write either "lazy" or "eager" for your answer.)

3. This is a problem about parameter passing mechanisms. Consider the following code in the defined language with static scoping, assignment, and lists.

```
let i = 1
    k = 5
in let g = proc (x, y, z)
              begin
               set x = +(y, z);
               set y = *(3, +(i,k));
               list(i, k, x, y, z)
             end
   in let res = (g i k +(i, 9))
      in cons(i, cons(k, res))
```

   (a) (10 points) What is the result of the above program if call-by-value is used as the parameter passing mechanism?

   (b) (10 points) What is the result of the above program if call-by-reference is used as the parameter passing mechanism?

   (c) (10 points) What is the result of the above program if call-by-value-result is used as the parameter passing mechanism? (Use left-to-right ordering if necessary.)

   (d) (10 points) What is the result of the above program if call-by-name is used as the parameter passing mechanism?

4. This is a problem about the object-oriented version of the defined language.

   (a) (15 points) What is the result of the following code?

```
class counter extends object
  field count
  method initialize() send self setCount(1)
  method setCount(i) set count = i
  method bump() send self setCount(add1(send self getCount()))
  method getCount() count
  method actionPerformed() send self bump()

class squareCounter extends counter
  method initialize() super initialize()
  method setCount(i) set count = *(i,i)

class publisher extends object
  field subscribers
  method initialize () set subscribers = list()
  method addSubscriber(o) set subscribers = cons(o, subscribers)
  method notify(subs)
    if null?(subs)
    then 1
    else begin send car(subs) actionPerformed();
               send self notify(cdr(subs))
         end
  method publish() begin send self notify(subscribers); 1 end

let ctr = new counter()
    sqCtr = new squareCounter()
    pub = new publisher()
    res = list()
in begin
    send pub addSubscriber(ctr);
    send pub addSubscriber(sqCtr);
    send pub publish();
    set res = list(send ctr getCount(), send sqCtr getCount());
    send pub publish();
    cons(send ctr getCount(), cons(send sqCtr getCount(), res))
   end
```

(b) (5 points) Consider the following program, where class `squareCounter` is as above.

```
% include classes counter and squareCounter here, as above
class doubleSquareCounter extends squareCounter
  method bump() begin send self bump(); send self bump() end

let dsc = new doubleSquareCounter()
in begin send dsc bump(); send dsc getCount() end
```

Briefly describe what happens when the above program is run.

The following just contains reference material for problems on later pages.

Types of helpers from the chapter 5 interpreters used on this test. These ADTs correspond to those in section 5.4.4 of the text.

```
eopl:error : (-> (symbol string datum ...) poof)

;; ---- ProcVal (procedure values) ADT --------
procval? : (type-predicate-for procval)
closure : (-> ((list-of symbol) expression environment) procval)
apply-procval : (-> (procval (list-of Expressed-Value)) Expressed-Value)

;; ---- reference ADT --------
a-ref : (forall (T) (-> (number (vector-of T)) (ref-of T)))
deref : (forall (T) (-> ((ref-of T)) T))
setref! : (forall (T) (-> ((ref-of T) T) void))
```

This page continues the reference material from the previous page. It describes types of procedures from the 5-4-4.scm interpreter that you can use for problems on later pages.

```
;; ---- environment ADT ----------
;; type predicate
environment? : (type-predicate-for environment)
;; constructors
empty-env : (-> () environment)
extend-env : (-> ((list-of symbol) (list-of Expressed-Value) environment)
                 environment)
extend-env-recursively : (-> ((list-of symbol) (list-of (list-of symbol))
                              (list-of expression) environment)
                             environment)
;; observers
apply-env : (-> (environment symbol) Expressed-Value)
apply-env-ref : (-> (environment symbol) (ref-of Expressed-Value))
defined-in-env? : (-> (environment symbol) boolean)

;; ----- Objects -------------------
(deftype object->fields (-> (object) (vector-of Expressed-Value)))
(deftype object->field-ids (-> (object) (list-of symbol)))
(deftype object->class-name (-> (object) symbol))
(deftype object->class (-> (object) class))

;; ----- The Class Table (Class Names) and Classes ----
(deftype class-name->field-ids (-> (symbol) (list-of symbol)))
(deftype class-name->methods (-> (symbol) (list-of method)))
(deftype class-name->super-name (-> (symbol) symbol))
(deftype class-name->field-length (-> (symbol) number))
(deftype class->class-name (-> (class) symbol))
(deftype class->super-name (-> (class) symbol))

;; ---- Expressed-Value ADT --------
;; upcasts
(deftype number->expressed (-> (number) Expressed-Value))
(deftype procval->expressed (-> (ProcVal) Expressed-Value))
(deftype object->expressed (-> (object) Expressed-Value))
(deftype list->expressed (-> ((list-of Expressed-Value)) Expressed-Value))
(deftype symbol->expressed (-> (symbol) Expressed-Value))
;; downcasts
(deftype expressed->number (-> (Expressed-Value) number))
(deftype expressed->procval (-> (Expressed-Value) ProcVal))
(deftype expressed->object (-> (Expressed-Value) object))
(deftype expressed->list (-> (Expressed-Value) (list-of Expressed-Value)))
(deftype expressed->symbol (-> (Expressed-Value) symbol))
;; tests
(deftype procval->expressed? (-> (Expressed-Value) boolean))
(deftype object->expressed? (-> (Expressed-Value) boolean))
```

5. (25 points) Consider the following new syntax in the object-oriented language.

   ⟨expression⟩ ::= `checkedassign` ⟨identifier⟩ `:` ⟨identifier⟩ `=` ⟨expression⟩

   The meaning of a checked assignment expression of the form `checkedassign` x `:` C `=` E is that the expression E is evaluated and if it evaluates to an object whose class is C or a subclass of C, then that object is assigned to x, and the value of E is returned; otherwise (if the value of E is not an object of a C or a subclass of C) an error is issued. The following are examples.

```
--> class c1 extends object
      method initialize() 1
    class c2 extends c1
    class c3 extends c2
    class notrelated extends object
      method initialize() 2
    let o1 = new c1()
        o2 = new c2()
        o3 = new c3()
        y = 3
    in begin
         checkedassign y : c1 = new c1();
         checkedassign y : c1 = o2;
         checkedassign y : c1 = o3;
         checkedassign o2 : c2 = o2;
         checkedassign o2 : c3 = o3;
         checkedassign o3 : c3 = checkedassign o3 : c3 = new c3();
         quote thatallworks
       end

thatallworks

--> class c1 extends object
      method initialize() 1
    class c2 extends c1
    class c3 extends c2
    let v = 4
    in checkedassign v : c3 = new c1()

Error reported by eval-expression: Right hand side is not an instance of class c3

--> class a extends object method initialize() 1
    class b extends object method initialize() 1
    let v = 4 in checkedassign v : b = new a()

Error reported by eval-expression: Right hand side is not an instance of class b
```

   Your job is to complete the implementation of the `checkedassign` expression, on the next page.

To start the implementation of the `checkedassign` expression, assume that the following are the new concrete and abstract syntax.

```
(define the-grammar
    ;; ...
    (expression ("checkedassign" identifier ":" identifier "=" expression)
       checkedassign-exp) )

(define-datatype expression expression?
  ;; ...
  (checkedassign-exp (id symbol?) (class-name symbol?) (rhs-exp expression?)) )
```

Your job is to fill in the code for `eval-expression` below, plus any helping procedures you need. (You can assume the other cases of `eval-expression` are already done. See the types of the built-in procedures you can use starting on page 5. There is more space for your answer on the next page if you need to continue.)

```
(deftype eval-expression (-> (expression environment) Expressed-Value))
(define eval-expression
  (lambda (exp env)
    (cases expression exp
      ;; ... assume other cases are done
      ;; fill in your answer for checkedassign below
```