cs1120: Exam 2

Out: 18 November
Due: 10:01am, Monday 23 November

Name:

UVa Email ID:

Directions

Work alone. You may not discuss these problems or anything related to the material covered by this exam with anyone except for the course staff between receiving this exam and class Wednesday.

Open resources. You may use any books you want, lecture notes, slides, your notes, and problem sets. You may use DrScheme and Python and any other language interpreter you want. You may also use external non-human sources including books and web sites. If you use anything other than the course book, slides, notes, and interpreters cite what you used. **You may not obtain any help from other humans** except for the course staff (who will only answer clarification questions).

Answer well. Answer as many of questions 1-10 as you can and the optional, ungraded question on the last page. A "full credit" answer for each question is worth 10 points (but it is possible to get more than 10 points for especially elegant and insightful answers). You may either: (1) write your answers on this exam or (2) type and write your answers into the provided Word template:
http://www.cs.virginia.edu/cs1120/exams/exam2/exam2.doc.

Whichever you choose, you must turn in your answers printed on paper and they must be clear enough for us to read and understand.

You should not need more space than is provided in the marked boxes to write good answers, but if you want more space you may use the backs or attach extra sheets. If you do, make sure the answers are clearly marked.

The questions are not necessarily in order of increasing difficulty. There is no time limit on this exam, but it should not take a well-prepared student more than two hours to complete. It may take you longer, though, so please do not delay starting the exam.

Full credit depends on the clarity and elegance of your answer, not just correctness. Your answers should be as short and simple as possible, but not simpler. Your programs will be judged for correctness, clarity and elegance, but you will not lose points for trivial errors (such as missing a closing parenthesis).

Pledge:

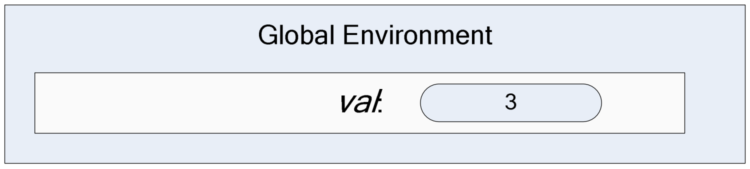
Scores:

Problem	Score	Notes
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
Total		

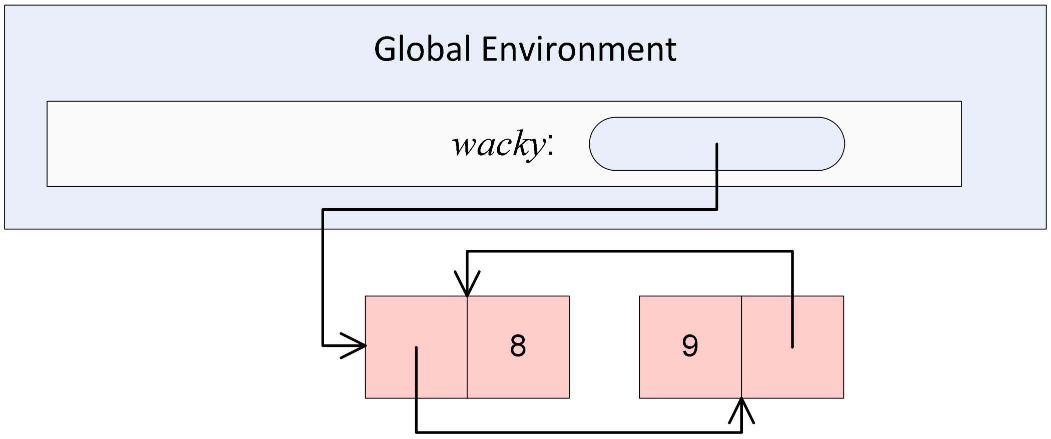
State and Mutation

For questions 1-3, each picture shows an environment resulting from evaluating one or more Scheme definitions and expressions. For each part, answer by providing Scheme code that produces the environment shown, starting from the initial global environment. There are many possible correct answers, but you should aim to make your code as simple as possible.

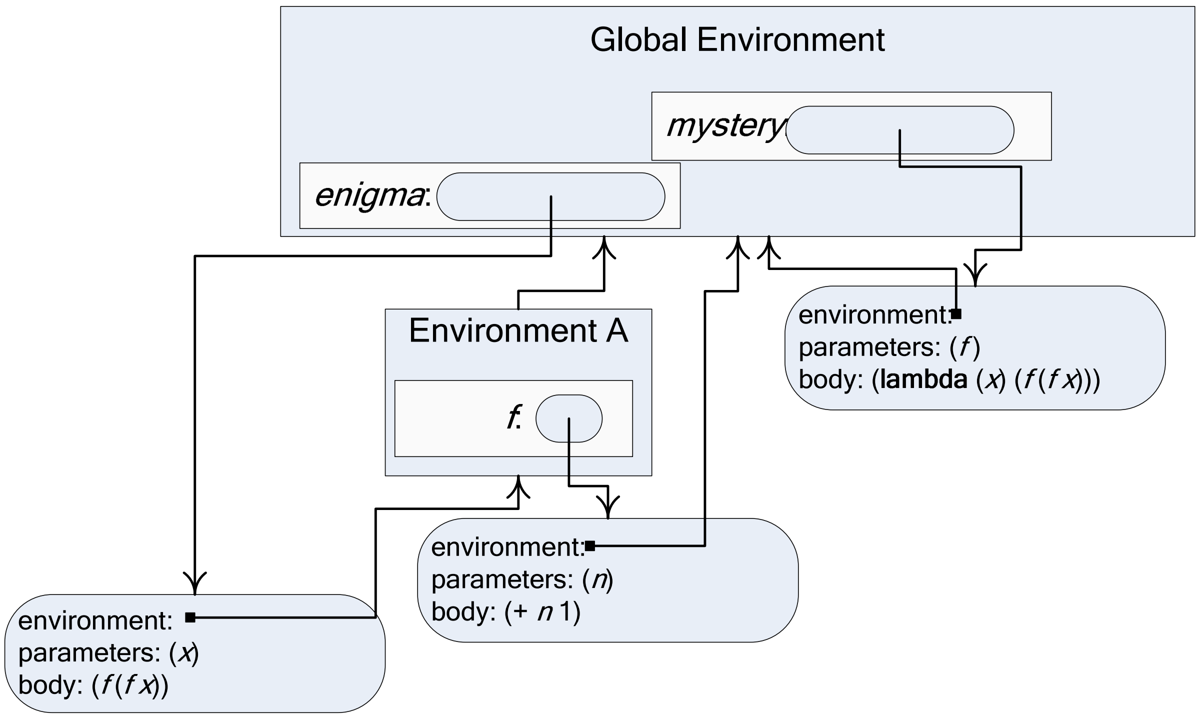
1.



2.



3.


Global Environment
mystery:
enigma:
Environment A
f:
environment:
parameters: (f)
body: (**lambda** (x) (f (f x)))
environment:
parameters: (n)
body: (+ n 1)
environment:
parameters: (x)
body: (f (f x))

Space and Time Analysis

Consider the list-merge and mlist-merge! Scheme procedures below:

```
(define (list-merge p1 p2)
   (if (null? p1) null
       (cons (car p1)
              (cons (car p2)
                     (list-merge (cdr p1) (cdr p2))))))

(define (mlist-merge! m1 m2)
  (if (null? m1) (void)
     (let ((m1next (mcdr m1))
           (m2next (mcdr m2)))
       (set-mcdr! m2 (mcdr m1))
       (set-mcdr! m1 m2)
       (mlist-merge! m1next m2next))))
```

For list-merge, assume both inputs are immutable lists of length *N*.
For mlist-merge!, assume both inputs are mutable lists of length *N*.

4a. What is the asymptotic *running time* of **list-merge**?

4b. What is the asymptotic *running time* of **mlist-merge!**?

5a. What is the asymptotic *memory use* of **list-merge**?

5b. What is the asymptotic *memory use* of **mlist-merge!**?

6. Consider this Python procedure (from Class 27):

```
def histogram(text):
    d = {}
    words = text.split()
    for w in words:
        if w in d:
            d[w] = d[w] + 1
        else:
            d[w] = 1
    return d
```

What is the asymptotic running time for histogram? State carefully all your assumptions and define all the variables you use.

Objects and Inheritance

Cordelia Lear does not believe in inheritance. She has defined the following three Python classes:

```
class Spider:
   def number_of_legs(self):  return 8
   def weave(self):  print "Weaving a web."

class SalticidaeSpider:
   def number_of_legs(self):  return 8
   def weave(self):  print "Weaving a web."
   def jump(self): print "Jumping!"

class TarantulaSpider:
   def number_of_legs(self): return 8
   def weave(self):  print "Weaving a web."
   def bite(self): print "Biting!"
```

7. Convince Cordelia that inheritance can be useful by defining the Spider, SalticidaeSpider, and TarantulaSpider classes with the exact same behavior as her classes, but with as few lines of code as possible.

Interpreters

The Charme interpreter from PS7/Chapter 11 does not define the **set!** special form, indeed it provides no mutation operators. Suppose we want extend Charme to include the set! special form, as defined by the Scheme evaluation rule for assignment:

Evaluation Rule 7: Assignment. To evaluate an assignment,

(set! *Name Expression***)**

evaluate the Expression, and replace the value associated with the name with the value of the expression. An assignment has no value.

You may assume this **updateVariable(self, name, value)** method has been added to the **Environment** class:

```
def updateVariable(self, name, value):
  if self._frame.has_key(name):
    self._frame[name] = value
  elif (self._parent):
    self._parent.updateVariable(name, value)
  else:
    evalError('Undefined name: %s' % (name))
```

8. Define an **evalAssignment(expr, env)** procedure. It should take as inputs a parsed Charme expression, which you may assume is an assignment expression, and the evaluation environment.

Note: you may use Python and the ps7 code to test your procedure. But, it is not necessary or encouraged to do so. You will not lose points on this question for unimportant syntactic mistakes.

```
def evalAssignment(expr, env):
```

9. Suppose the assignment expression is also added to MemoCharme (the language you have at the end of Problem Set 7 where the results of procedure applications are memoized). Illustrate the problems this might cause by showing a sequence of Charme expressions that will evaluate to different values in the versions of Charme with and without memoization.

Sally Snake has grown fond of the iteration constructs provided by Python, and wants you to extend Charme to provide a while expression similar to the while statement in Python. Its evaluation rule is:

Evaluation Rule: While. To evaluate a while expression,

(while *Expression*$_{\text{pred}}$ *Expression*$_{\text{body}}$ *Expression*$_{\text{final}}$**)**

evaluate the predicate expression (*Expression*$_{\text{pred}}$). If it evaluates to a false value, the value of the while expression is the value of the final expression (*Expression*$_{\text{final}}$). Otherwise, evaluate the body expression (*Expression*$_{\text{body}}$).

For example, here is a Charme program using while (this also assumes the assignment expression from question 7):

```
Charme> (define loopy (lambda (n) (while (< n 10) (set! n (* n 2)) n)))
Charme> (loopy 3)
12
```

10. Define an evalWhile(expr, env) procedure that implements the evaluation rule for a while expression. It should take as inputs a parsed Charme expression, which you may assume is an assignment expression, and the evaluation environment.

11. (optional, ungraded) Do you feel your performance on this exam will fairly reflect your understanding of the course material so far? If not, explain why. (Feel free to use the back or additional pages to provide more comments as you see fit.)