

Name: _____

**CSE 341, Spring 2008, Final Examination
11 June 2008**

Please do not turn the page until everyone is ready.

Rules:

- The exam is closed-book, closed-note, except for one side of one 8.5x11in piece of paper.
- **Please stop promptly at 10:20.**
- You can rip apart the pages, but please staple them back together before you leave.
- There are **100 points** total, distributed **unevenly** among **7** questions (most with multiple parts).
- When writing code, style matters, but don't worry about indentation.

Advice:

- Read questions carefully. Understand a question before you start writing.
- **Write down thoughts and intermediate steps so you can get partial credit.**
- The questions are not necessarily in order of difficulty. **Skip around.**
- If you have questions, ask.
- Relax. You are here to learn.

Name: _____

1. (a) (10 points) Write a Scheme function `maxInt` that takes one argument.
 - Assume the argument is a “nested number list.” A “nested number list” is a list where each element is either a number or another “nested number list.” A “nested number list” can be the empty list.
 - Return the value of the largest number anywhere in the argument, or `#f` if the argument has no numbers (*one* such example is the empty list).
 - Sample solution is about 8 lines; this is just a rough guide.
- (b) (5 points) Give two reasons why a similar function in ML would not type-check.

Name: _____

2. (a) (7 points) Suppose this expression appears somewhere in a Scheme program:

```
(f (lambda (x y) (g x y)))
```

What is a simpler and equivalent expression that could always replace the expression above?

- (b) (5 points) Suppose this expression appears somewhere in a Scheme program:

```
(f (lambda (x y) ((g) x y)))
```

Consider your answer to part (a) except replace any `g` in it with `(g)`. Call this the “modified answer.” Replacing the expression above with the “modified answer” does *not* necessarily produce an equivalent program. Give an example that shows this. That is, write a program that uses `(f (lambda (x y) ((g) x y)))` somewhere and replacing this with the “modified answer” would change how the program behaves. Briefly explain your answer.

Name: _____

3. In this problem, we consider an interpreter for a language FUPL (for final useless programming language) which is like MUPL from Homework 5, but the language does not have functions. Instead it has `mylet` and `myletstar`, which are like Scheme's `let` and `let*`. Programs are built using these structs:

```
(define-struct var (string))          ;; a variable, e.g., (make-var "foo")
(define-struct int (num))              ;; a constant number, e.g., (make-int 17)
(define-struct add (e1 e2))           ;; add two expressions
(define-struct mylet (bindings body)) ;; a let expression
(define-struct myletstar (bindings body)) ;; a let* expression
```

For both forms of let-expressions, the `bindings` field is assumed to hold a Scheme list of Scheme pairs where each pair is a Scheme string (the variable name) in the `car` position and a FUPL expression in the `cdr` position. The `body` field is assumed to hold a FUPL expression.

Here is an interpreter with two blanks you will fill in. It uses `append`, `map`, and `fold`, which are defined below in the expected way. The questions are on the next page.

```
(define (envlookup env str)
  (cond [(null? env) (error "unbound variable during evaluation" str)]
        [(equal? (caar env) str) (cdar env)]
        [#t (envlookup (cdr env) str)]))

(define (eval-prog p)
  (letrec ([f (lambda (env p)
                (cond [(var? p) (envlookup env (var-string p))]
                      [(int? p) p]
                      [(add? p) (let ([v1 (f env (add-e1 p))]
                                       [v2 (f env (add-e2 p))])
                                  (if (and (int? v1) (int? v2))
                                      (make-int (+ (int-num v1) (int-num v2)))
                                      (error "FUPL addition applied to non-number"))))]
                      [(mylet? p) (f (append (map _____
                                              (mylet-bindings p))
                                              env)
                                       (mylet-body p))]
                      [(myletstar? p) (f (fold _____
                                              env
                                              (myletstar-bindings p))
                                       (myletstar-body p))]
                      [#t (error "bad FUPL expression")])])
    (f () p)))

(define (append lst1 lst2)
  (if (null? lst1)
      lst2
      (cons (car lst1) (append (cdr lst1) lst2))))

(define (map f lst)
  (if (null? lst)
      ()
      (cons (f (car lst)) (map f (cdr lst)))))

(define (fold f acc lst)
  (if (null? lst)
      acc
      (fold f (f acc (car lst)) (cdr lst))))
```

Name: _____

Problem 3 continued

- (a) (5 points) In English, briefly explain the difference between Scheme's `let` and `let*`.
- (b) Complete the interpreter as follows:
 - i. (5 points) Complete the `mylet` case by providing the first argument to `map`.
 - ii. (5 points) Complete the `myletstar` case by providing the first argument to `fold`.

Remember the environment is a mapping from strings to FUPL values, and we need to evaluate each FUPL expression in the correct environment. An English explanation of what you are trying to do may help with partial credit.

- (c) (5 points) Complete the FUPL program below such that evaluating it would produce `(make-int 2)` but changing the `make-myletstar` to `mylet` would make evaluating it produce `(make-int 1)`. Remember to use the FUPL constructors `make-int` and `make-var` in the right places. (The sample solution does not use `make-add`.)

```
(make-mylet (list (cons "a" (make-int 1)))
            (make-myletstar ...
                          ...))
```

Name: _____

4. (15 points)

This Ruby class makes a private copy of an array and has methods for updating the array and returning its sum:

```
class ArraySum
  def initialize arr
    @arr = Array.new arr
  end
  def sum
    @arr.inject {|acc,x| acc + x} # inject is the Ruby library's word for fold
  end
  def update(i,v)
    @arr[i] = v
  end
end
```

Write a subclass `MemoizedArraySum` that works as follows:

- An instance does not compute the array's sum until the `sum` method is called.
- If an instance's `sum` method is called again *before* the `update` method is called, then `sum` returns an answer that is already computed.
- The first time `sum` is called after `update` is called, the sum is simply recomputed (without using any precomputed information).

Use `super` as appropriate.

Name: _____

5. Because Ruby is dynamically typed, method calls can fail at run-time as in this example program:

```
class C
  def initialize
    m(3,4)
  end
end
C.new # needs C to have a method it does not have
```

Consider a “type system” for Ruby that is not intended to prevent *all* method-call errors, but is intended to prevent all errors on method calls to `self` (for example the call to `m` above). Suppose the “type system” works as follows:

- We first look at the whole program to gather all the methods defined by each class and mixin.
 - Then for every class, we look at the body of every method defined in the class. If a method body in some class `C` contains a call to a method `m`, then we require one or more of the following:
 - Class `C` defines a method `m`.
 - A superclass of `C` defines a method `m`.
 - `C` or a superclass of `C` includes a mixin with a method `m`.
- (a) (7 points) Our “type system” rightly rejects the program above. However, there are other programs that:
- Have exactly the same definition for class `C`
 - When run, execute the `initialize` method in class `C`.
 - Do not have a run-time error.

Give such a program. Hint: Use `new`, but not on class `C`.

- (b) (4 points) Does part (a) demonstrate that our “type system” is unsound or that it is incomplete?
- (c) (3 points) Our “type system” is in fact unsound and incomplete given that we want every call to `self` to “make sense” and execute some method body. Give a reason the type system is unsound (if you answered “incomplete” for part (b)) or incomplete (if you answered “unsound” for part (b)).

Name: _____

6. For this problem, short answers (about 2 sentences) can be sufficient. That is, we are not asking for a long essay.
- (a) **(6 points)** Why are Ruby mixins more powerful than Java-style interfaces?
 - (b) **(6 points)** Even if Ruby did not have mixins, why would it not be useful to add Java-style interfaces to Ruby.

Name: _____

7. Suppose we have a Java-like statically typed object-oriented language where (unlike actual Java) a method in a subclass with the same name always overrides a same-named method in a superclass. Consider this code skeleton:

```
class A1 extends Object { int x() { return 1; } }
class A2 extends A1     { int y() { return 2; } }
class A3 extends A2     { int z() { return 3; } }

class C2 extends Object {
    void m(A2 a) { a.y(); }
}
class C1 extends C2 {
    void m(A1 a) { a.x(); }
}
class C3 extends C2 {
    void m(A3 a) { a.z(); }
}
class Main {
    public static void main(String [] args) { ... }
}
```

- (a) (6 points) Which subclass of C2 should be rejected in order to keep the type system sound?
- (b) (6 points) Fill in the ... above such that if the subclassing you chose in part (a) were allowed, then the whole program would type-check, but at run-time there would be a message-not-understood error. Explain what method call would cause this error. (As in Java, write `new C()` to make a new object of class C. Also as in Java, assume objects can be cast to supertypes.)