CSE 341 - Programming Languages
Final exam - Winter 2010 - Answer Key

1. (10 points) Write a Haskell function `until_false` that takes a function `f` and a list `xs`. It returns a list of the elements of `xs` for which `f` evaluates to true — but as soon as `f` evaluates to false, it stops and doesn't examine any more elements. Here are some examples:

```
until_false even [2,4,6,7,10,12]  =>  [2,4,6]
until_false even [2..]  =>  [2]
until_false even []  =>  []

until_false f [] = []
until_false f (x:xs) | f x       = x : until_false f xs
                     | otherwise = []
```

2. (5 points) What is the most general possible type for your `until_false` function from Question 1?

```
until_false :: (a -> Bool) -> [a] -> [a]
```

3. (20 points) The original Scheme metacircular interpreter doesn't include `or`.

   (a) Write a Scheme function `or->combination` that implements `or` as a derived expression. Assume that the argument to `or->combination` is a legal `or` expression.

   This is a bit tricky. If the `or` has one or more expressions, you need to evaluate the first one and save the value (using a temporary variable), to avoid evaluating it more than once. Then test the temporary variable. Further, you want to avoid having a temporary name override some other name used in the expressions — to accomplish this, the code below uses `gensym` to generate a new, unique name.

```
(define (or->combination expr)
  (cond ((null? (cdr expr)) '#f)  ; no arguments to 'or'
     ((null? (cddr expr)) (cadr expr))  ; one argument to 'or'
     (else ; more than one argument
      (let ((var (gensym)))
        (list 'let (list (list var (cadr expr)))
              (list 'if var var (cons 'or (cddr expr))))))))
```

   Here is another version, using quasi-quoting:

```
(define (or->combination2 expr)
 (cond ((null? (cdr expr)) '#f)  ; no arguments to 'or'
    ((null? (cddr expr)) (cadr expr))  ; one argument to 'or'
    (else ; more than one argument
     (let ((var (gensym)))
       `(let ((,var ,(cadr expr)))
           (if ,var ,var (or ,@(cddr expr)))))))))
```

   (b) Using your definition, what is the value of the following expressions?

   i. `(or->combination '(or))`

      `#f`

   ii. `(or->combination '(or (f 10)))`

      `(f 10)`

   iii. `(or->combination '(or (= x 5) (< y 10) (member x xs)))`

```
(let ((g2 (= x 5)))
  (if g2 g2 (or (< y 10) (member x xs))))
```

(There should have been a space between the < and the y – fixed in this answer key.)

Hints: remember that for derived expressions, the Scheme interpreter takes a list representing the expression in one form, in this case as a list `(or ...)`, and returns another list representing an equivalent expression that either doesn't use `or`, or that uses a simpler `or` expression (i.e., a recursive case). You can use `if` or `cond` in your equivalent expression. Also remember the semantics of `or`: expressions are evaluated left to right, and the value of the first expression that evaluates to something other than `#f` is returned. Any remaining expressions are not evaluated. If all the expressions evaluate to `#f`, return `#f`. If there are no expressions, return `#f`. (Never evaluate an expression more than once, of course.)

If you need a brand-new symbol, you can generate one using `(gensym)`.

4. (10 points)

   (a) Write one or more rules to define the `positives` constraint in CLP(R). `positives` takes two arguments. It should succeed if both arguments are lists of numbers, and the second argument contains all the positive numbers in the first list (in the same order as in the first list), and only positive numbers. Don't use cut in your rules. For example, these goals should succeed:

```
positives([4,50,-1,-10,2], [4,50,2]).
positives([], []).
```

   and these should fail:

```
positives([4,50,-1,-10,2], [4,50]).
positives([4,50,-1,2], [4,50,-1]).
positives([], [10]).
```

   Solution:

```
positives([],[]).
positives([X|Xs],[X|Ys]) :- X>0, positives(Xs,Ys).
positives([X|Xs],Ys) :- X<=0, positives(Xs,Ys).
```

   (b) What are all of the answers returned for the following goals, using your rules? (If there are infinitely many, give the first 3.)

   - `positives([3,-4,0], B).`
     `B = [3]`
   - `positives([3,10], B).`
     `B = [3, 10]`
   - `positives(A, []).`

```
A = []

A = [_t1]
_t1 <= 0

A = [_t1, _t2]
_t1 <= 0
_t2 <= 0
```

5. (10 points) Consider the `member` rule in CLP(R):

```
member(X,[X|Xs]).
member(X,[_|Xs]) :- member(X,Xs).
```

Draw the simplified derivation trees for the following goals. If the tree is infinite, say that, and include at least the first 3 answers.

(a) `member(A,[2,3]).`

(b) `member(2,As).`

See separate scanned answer.

6. (8 points) Consider a version of the CLP(R) `member` rule with a cut:

```
member_cut(X,[X|Xs]) :- !.
member_cut(X,[_|Xs]) :- member_cut(X,Xs).
```

What are all of the answers returned for the following goals? If there are an infinite number of answers, give the first three.

```
?- member_cut(10,[1,2,3]).
no.

?- member_cut(A,[1,2,3]).
A=1.

?- member_cut(A,[]).
no.

?- member_cut(10,As).
As=[10].
```

7. (6 points) Consider the following example in an Algol-like language.

```
begin
integer n;
procedure p(j: integer);
    begin
    j := j+n;
    n := 2*n+j;
    print(n);
    print(j);
    end;
n := 10;
p(n);
print(n);
end;
```

(There should have been a : in the assignment to j - fixed above.)

(a) What is the output when j is passed by value?
40 20 40

(b) What is the output when j is passed by value result?
40 20 20

(c) What is the output when j is passed by reference?
60 60 60

8. (10 points) In Ruby, `true` and `false` are objects: `true` is an instance of `TrueClass`, and `false` is an instance of `FalseClass`. `and`, `or`, and `!` (not) are built-in, but they could actually be user-defined methods. Write methods for `and`, `and_sc`, `or`, `or_sc`, and `not` for both `TrueClass` and `FalseClass` (so 10 methods in all). `and` should always evaluate its argument, while `and_sc` should do short-circuit evaluation (so it should take a block rather than a regular argument); and similarly for `or` and `or_sc`. Here some example expressions and the result of evaluating them:

```
(1==2).not  =>  true
(1==2).and(1/0==1)  =>  ZeroDivisionError exception
(1==2).and_sc {1/0==1}  =>  false
```

```
class TrueClass
  def and(b)
    b
  end
  def and_sc
    # return the value of the block
    yield
  end
  def or(b)
    true
  end
  def or_sc
    # this takes a block, but we ignore it
    true
  end
  def not
    false
  end
end

class FalseClass
  def and(b)
    false
  end
  def and_sc
    false
  end
  def or(b)
    b
  end
  def or_sc
    yield
  end
  def not
    true
  end
end
```

9. (8 points) Consider the following Java programs `Test1`, `Test2`, `Test3`, and `Test4`. In each case, does the program compile correctly? If so, does it execute without error, or is there an exception?

```
/***************** Test1 *****************/
import java.awt.Point;
class Test1 {
    public static void main(String[] args) {
        Point[] a;
        a = new Point[10];
        a[0] = new Point(10,20);
        test(a);
    }
    public static void test(Object[] b) {
        b[1] = "clam";
```

```
    }
}

gets a runtime ArrayStoreException
/*****************************************/


/***************** Test2 *****************/
import java.awt.Point;
import java.util.LinkedList;
class Test2 {
    public static void main(String[] args) {
        LinkedList<Point> a = new LinkedList<Point>();
        a.add(new Point(10,20));
        test(a);
    }

    public static void test(LinkedList<?> b) {
        b.add(new Point(20,30));
    }
}

gets a compile-time error
/*****************************************/


/***************** Test3 *****************/
import java.awt.Point;
import java.util.LinkedList;
class Test3 {
    public static void main(String[] args) {
        LinkedList<Point> a = new LinkedList<Point>();
        a.add(new Point(10,20));
        test(a);
    }

    public static void test(LinkedList<?> b) {
        b.add(null);
    }
}

executes without error
/*****************************************/


/***************** Test4 *****************/
import java.awt.Point;
import java.util.LinkedList;
class Test4 {
    public static void main(String[] args) {
        LinkedList<Point> a = new LinkedList<Point>();
```

```
        a.add(new Point(10,20));
        test(a);
    }

    public static void test(LinkedList<Object> b) {
        b.add(new Point(30,40));
    }
}

gets a compile-time error
/**************************************/
```

10. (5 points) What is the result of evaluating the following Ruby expressions? (Hint: String is a subclass of Object.)

    (a) `"squid".class => String`

    (b) `"squid".class.class => Class`

    (c) `"squid".class.class.class => Class`

    (d) `"squid".class.superclass => Object`

    (e) `"squid".class.superclass.superclass => nil`

11. (4 points) Consider the following Scheme program.

```
(define n 10)
(define (squid n)
   (clam n))
(define (clam k)
   (+ n k))
```

    (a) What is the result of evaluating `(squid 0)`? **10**

    (b) Suppose that Scheme used dynamic rather than lexical scoping. In that case what would be the result of evaluating `(squid 0)`? **0**

12. (10 points) Write a `count` method for the `Enumerable` mixin in Ruby. The `count` method should take any object as a parameter, and return the number of occurences of that object in the collection. (What counts as an occurrence should be determined by the == test). For example, `["clam", "squid", "clam"].count("clam")` should evaluate to 2.

```
module Enumerable
  def count(x)
    n = 0
    self.each {|i| n=n+1 if x==i}
    return n
  end
end
```