CSE 341 - Programming Languages
Final exam - Autumn 2012 - Answer Key

Total points possible: 134

1. (10 points) Write a Haskell function `mapfns`, and a type declaration for it (the most general type possible). `mapfns` takes a list of functions and a list of items, and that returns a new list consisting of the result of applying each function to the corresponding item. If there are a different number of functions and items, the resulting list is the same length as the shorter of the two. Examples:

```
mapfns [(+1), (*2), sqrt] [2.0, 3.5, 4.0]  => [3.0, 7.0, 2.0]
mapfns [(+1), (+1)] [1..]  => [2, 3]

mapfns :: [t -> a] -> [t] -> [a]

mapfns (f:fs) (x:xs) = f x : mapfns fs xs
mapfns _ _ = []
```

2. (10 points) Write a Racket function `twice` that takes a list as an argument, and returns `#t` if there is at least one adjacent repeated element in the list, and otherwise `#f`. You can assume that the argument is a proper list. Use `equal?` to test whether one element is a repeat of the previous one. Examples:

```
(twice '())  => #f
(twice '(a b b c d e))  => #t
(twice '(a b c a b))  => #f


(define (twice s)
  (cond ((null? s) #f)
        ((null? (cdr s)) #f)
        ((equal? (car s) (cadr s)) #t)
        (else (twice (cdr s)))))
```

3. (8 points) Consider the `my-or` macro discussed in class. This works exactly the same as the built-in `or` macro in Racket.

```
(define-syntax my-or
  (syntax-rules ()
    ((my-or) #f)
    ((my-or e) e)
    ((my-or e1 e2 ...)
     (let ((temp e1))
       (if temp
           temp
           (my-or e2 ...))))))
```

   It relies on macro hygiene. Demonstrate this by writing Racket code that would result in a wrong answer from `my-or` if Racket didn't have macro hygiene. Also say both what the correct answer is, and what the answer would be if Racket didn't have macro hygiene.

```
(let ((temp 3))
  (or #f (= temp 3)))
```

Without macro hygiene, when the macro is expanded it uses a variable temp in the if expression. This will have the value #f, which shadows the binding of temp to 3. The expression should have the value #t,but without macro hygiene it would evaluate to #f.

4. (8 points) Is MUPL statically typed? If it is, give a MUPL expression that has a type error that would be detected statically. If it is not, just say "no". (For both this and the following question, you don't need to remember the exact MUPL structs to build MUPL expressions; but if you're not sure you've got the names correct say what they are and what they do so that we can grade these.)

   No.

   Is MUPL type-safe? If it is, give a MUPL expression that has a type error that would be detected dynamically. If it is not, give a MUPL expression that would be evaluated incorrectly due to the type error.

   As in the assignment, you should assume MUPL programs are syntactically correct, e.g., do not worry about wrong things like `(int "hi")` or `(int (int 37))`. Think of `(int 3)` as part of the program's syntax its how you write the number 3 in MUPL.

   Yes, MUPL is type-safe. The assignment required including runtime type checking in functions. For example, this expression would give a runtime MUPL error:

   ```
   (fst (int 3))
   ```

   The function fst gets the first element of a list, but we're giving it an integer argument.

5. (10 points) Consider a `twice` rule in Prolog, which looks for an element that occurs twice in a row in a list:

   ```
   twice([X,X|Xs],X).
   twice([X|Xs],Y) :- twice(Xs,Y).
   ```

   What are all the answers returned for the following goals? If there are an infinite number, say that, and include at least the first 3 answers.

   (a) `twice([1,2,2],B).` Answer: `B=2.`

   (b) `twice([1,2,2,1],B).` Answer: `B=2.`

   (c) `twice([1,2,3,2,1],B).` Answer: `false.`

   (d) `twice([1,2,2,2,3,1,1],B).` Answers: `B=2; B=2; B=1`

   (e) `twice(As,1).` Answers: `As = [1, 1|X1];`
   `As = [X1, 1, 1|X2] ; As = [X1, X2, 1, 1|X3]; ....` (infinite number of answers – here `X1` etc are going to be variable names that Prolog generates)

6. (10 points) Now consider the `twice` rule again, but with cut:

   ```
   twice([X,X|Xs],X) :- !.
   twice([X|Xs],Y) :- twice(Xs,Y).
   ```

   What are all the answers returned for the following goals? (These are the same as for Question 5.) If there are an infinite number, say that, and include at least the first 3 answers.

   (a) `twice([1,2,2],B).` Answer: `B=2.`

   (b) `twice([1,2,2,1],B).` Answer: `B=2.`

   (c) `twice([1,2,3,2,1],B).` Answer: `false.`

(d) `twice([1,2,2,2,3,1,1],B).  Answer: B=2.`

(e) `twice(As,1). Answer: As = [1, 1|X1].`

7. (6 points) Which of the following lists represent valid difference lists? For valid difference lists, what list do they represent?

```
[a,b,c]\[c]
valid: represents [a,b]

[a,b,c]\[a,b,c]
valid: represents []

[a,b,c]\[a,b]
not valid

[a,b,c|T]\T
valid: represents [a,b,c]

[a,b,c|T]\[b,c|T]
valid: represents [a]

T\T
valid: represents []
```

8. (12 points) This question asks you to write some Prolog facts and rules — but as a twist, the facts and rules concern inheritance in an object-oriented langage.

   (a) Write Prolog facts to represent that the class `octopus` is a subclass of `seacreature`, `seacreature` is a subclass of `animal`, and `animal` is a subclass of `object`. (Since we don't want Prolog to think your names are Prolog variables, we're using lower-case letters for the class names.)

```
subclass(octopus,seacreature).
subclass(seacreature,animal).
subclass(animal,object).
```

   (b) Write one or more Prolog rules for the `ancestor` relation. This is defined recursively: a class is an ancestor of itself, and also its superclass, the superclass of its superclass, etc. So the goal `ancestor(seacreature,X)` should succeed with `X=seacreature`. If you backtrack, it should succeed with `X=animal`, then `X=object`, and then fail. For full credit you need to produce the answers in exactly that order.

```
ancestor(A,A).
ancestor(A,C) :- subclass(A,B), ancestor(B,C).
```

   (c) Now suppose that our object-oriented language has multiple inheritance. Add rules for two more classes: `dolphin` and `mammal`, with `dolphin` a subclass of both `mammal` and `seacreature`, and `mammal` a subclass of `animal`. Do you need to modify your `ancestor` rule(s) to make them work with multiple inheritance? If so, write down the new version here; if not, say that no modification is needed.

```
subclass(dolphin,mammal).
subclass(mammal,animal).
subclass(dolphin,seacreature).

No modification is needed for the ancestor rules.
```

What are all the answers are returned for the goal `ancestor(dolphin,X)` ? For full credit you need to write down the answers in exactly the order that your program will return them. The answer `X=dolphin` should be produced first, but after that you can return the answers in any order; duplicates are OK.

```
?- ancestor(dolphin,X).
X = dolphin ;
X = mammal ;
X = animal ;
X = object ;
X = seacreature ;
X = animal ;
X = object ;
false.
```

9. (10 points) Consider the difference list version of `reverse` as given in the lecture notes:

```
reverse(Xs,Rs) :- reverse_dl(Xs,Rs\[]).

reverse_dl([],T\T).
reverse_dl([X|Xs],Rs\T) :- reverse_dl(Xs,Rs\[X|T]).
```

Draw the derivation tree for the following goal:

```
?- reverse([a,b,c],R).
```

Please see the scan of the hand-drawn tree at the end of this pdf.

10. (12 points) Consider the following Ruby classes and mixins.

```
class C1
  def print_me
    "C1 print_me"
  end
  def test
    1
  end
end

module M1
  def print_me
    "M1 print_me"
  end
  def test
    10+super
  end
end

module M2
  def print_me
    "M2 print_me"
  end
  def test
```

```
    100
  end
end

class C2 < C1
  include M1
end

class C3 < C1
  include M1, M2
end
```

Now define variables c1, c2, and c3 as follows:

```
  c1 = C1.new
  c2 = C2.new
  c3 = C3.new
```

What is printed as a result of evaluating the following expressions? (Remember that the value returned by a method is the value of the last expression, if there isn't an explicit return.)

```
  c1.print_me
 => "C1 print_me"

  c2.print_me
 => "M1 print_me"

  c3.print_me
 => "M1 print_me"

  c1.test
 => 1

  c2.test
 => 11

  c3.test
 => 110
```

11. (16 points) Define a Ruby class `MySet` that represents a set. (There's already a built-in class `Set` — define your own however, rather than using the built-in one.) `MySet` should be a subclass of `Object` and should mix in `Enumerable`. Internally, it should use a Ruby array to hold its elements. As a reminder, a set is an unordered collection of elements, with no duplicates. The union of two sets A and B is a new set consisting of all elements that are in either A or B. The intersection of two sets A and B is a new set consisting of the elements that are in both A and B.

    `MySet` should define the following methods:

    `initialize` Initialize this to be an empty set.

    `add(element)` Add an element to this set, if it's not already a member. (This changes the set.)

    `union(other)` Return a new set that is the union of the receiver and `other`. This doesn't change the set (no side effect).

`intersect(other)` Return a new set that is the intersection of the receiver and `other`. This doesn't change the set (no side effect).

`each` Needed for the `Enumerable` mixin.

```
# for some reason Ruby doesn't like the name MySet!

class TestMySet
  include Enumerable

  def initialize
    @contents = []
  end

  def each (&blk)
    @contents.each(&blk)
  # lots of variations are possible to handle the block ...
  end

  def add(e)
    if not( @contents.include?(e))
      @contents.push(e)
    end
  end

  def union(s)
    ans = TestMySet.new()
    each {|e| ans.add(e)}
    s.each {|e| ans.add(e)}
    ans
  end

  def intersect(s)
    ans = TestMySet.new()
    s.each {|e| if @contents.include?(e) then ans.add(e) end}
    ans
  end

end
```

12. (12 points) Consider the following Java programs `Test1`, `Test2`, and `Test3`. In each case, does the program compile correctly? If it doesn't compile, indicate all the lines that result in a compilation error and what the error is. If it does compile, does it execute without error, or is there an exception? If it executes, what is the output, either for the whole program or up until the point where there is an exception? (Except for possible type errors, the programs are legal Java.)

```
/***************** Test1 *****************/
class Test1 {
    public static void main(String[] args) {
        String[] a;
        Object[] b;
```

```
        a = new String[5];
        b = a;
        a[0] = "oyster";
        test(a,1);
        test(b,2);
        test( (Object[]) a, 3);
        test( (String[]) b, 4);
        for(int i = 0; i<a.length; i++) {
            System.out.print(a[i]);
            System.out.print(" ");
        }
        System.out.print("\n");
    }
    public static void test(Object[] c, int i) {
        c[i] = "clam";
    }
    public static void test(String[] c, int i) {
        c[i] = "squid";
    }
}
```

Executes without error - output is:
oyster squid clam clam squid

```
/***************** Test2 *****************/
import java.util.LinkedList;
class Test2 {
    public static void main(String[] args) {
        LinkedList<String> a;
        LinkedList<Object> b;
        a = new LinkedList<String>();
        b = a;
        a.addFirst("oyster");
        test(a,1);
        test(b,2);
        for (String s : a) {
            System.out.print(s);
            System.out.print(" ");
        }
        System.out.print("\n");
    }
    public static void test(LinkedList<Object> c, int i) {
        c.add(i,"clam");
    }
    public static void test(LinkedList<String> c, int i) {
        c.add(i,"clam");
    }
}
```

Doesn't compile - there are type errors:

```
Test2.java:21: test(java.util.LinkedList<java.lang.Object>,int) is
already defined in Test2
    public static void test(LinkedList<String> c, int i) {
                        ^
Test2.java:8: incompatible types
found   : java.util.LinkedList<java.lang.String>
required: java.util.LinkedList<java.lang.Object>
        b = a;
            ^
Test2.java:10: test(java.util.LinkedList<java.lang.Object>,int) in
Test2 cannot be applied to
(java.util.LinkedList<java.lang.String>,int)
        test(a,1);
        ^
3 errors
```

These errors are all because LinkedList<String> is not a subtype of LinkedList<Object> .

```
/****************************************/
class Test3 {
    public static void main(String[] args) {
        String[] a;
        Object[] b;
        a = new String[2];
        b = a;
        a[0] = "oyster";
        System.out.println("added an oyster");
        b[1] = new Integer(5);
        System.out.println("added an integer");
        for(int i = 0; i<a.length; i++) {
            System.out.print(a[i]);
            System.out.print(" ");
        }
        System.out.print("\n");
    }
}
```

Compiles, but gets a runtime exception:

```
added an oyster
Exception in thread "main" java.lang.ArrayStoreException:
java.lang.Integer
at Test3.main(Test3.java:9)
```

13. (10 points) True or false?

    (a) A Haskell expression of type `IO t` can never occur inside another expression of type `(Num t) => [t]`

in a expression that typechecks correctly.
`True`

(b) In Java, adding an upcast can never change whether or not a program compiles, but could change the behavior of a program that does compile without without the upcast.
`False (the upcast could cause a type error at compile time)`

(c) In Java, `Point[]` is a subtype of `Object[]`.
`True`

(d) In Java, `ArrayList<Point>` is a subtype of `ArrayList<Object>`.
`False`

(e) In Java, `ArrayList<Point>` is a subtype of `ArrayList<?>`.
`True`

(f) In Ruby, class `Object` is an instance of itself.
`False`

(g) In Ruby, class `Object` is a subclass of itself.
`False`

(h) In Ruby, class `Class` is an instance of itself.
`True`

(i) In Ruby, class `Class` is a subclass of itself.
`False`

(j) A Ruby class can have multiple superclasses, but only one mixin.
`False`

9. (10 points) Consider the difference list version of `reverse` as given in the lecture notes:

```
R1  reverse(Xs,Rs) :- reverse_dl(Xs,Rs\[]).
R2  reverse_dl([],T\T).
R3  reverse_dl([X|Xs],Rs\T) :- reverse_dl(Xs,Rs\[X|T]).
```

Draw the derivation tree for the following goal:

```
?- reverse([a,b,c],R).
```

/ R1
reverse_dl([a,b,c], R\[]).

/ R3
reverse_dl([b,c], R\[a]).

/ R3
reverse_dl([c], R\[b,a]).

/ R3
reverse_dl([], R\[c,b,a]).

/ R2
true R = [c,b,a]