1. (10 points) Write a Haskell function \texttt{union}, and a type declaration for it (the most general type possible). \texttt{union} takes two lists (representing sets), and that returns a new list consisting of the set-theoretic union of them. The order of the values in the returned list doesn’t matter. You can assume the lists don’t contain any duplicate items. Examples:

\texttt{union [1,2,3] []} \Rightarrow [1,2,3]
\texttt{union [1,10,3] [10,20,3]} \Rightarrow [1,10,3,20]
2. (10 points) Consider a `starts_with` rule in Prolog, which succeeds if the two arguments are lists, and the second argument starts with the elements in the first argument perhaps plus some more:

\[
\begin{align*}
\text{starts} \_ \text{with}([ ], _) & \text{.} \\
\text{starts} \_ \text{with}([X|Xs],[X|Ys]) & :- \text{ starts} \_ \text{with}(Xs,Ys). \\
\end{align*}
\]

For example, \( \text{starts} \_ \text{with}([1,2,3],[1,2,3,4,5]) \) succeeds. \( \text{starts} \_ \text{with}([1],[2,3,4,5]) \) fails.

What are all the answers returned for the following goals? If there are an infinite number, say that, and include at least the first 4 answers. (If you need to create a new variable, write it as \_G14 and so forth — the exact number after the \( G \) doesn’t matter.)

(a) \( \text{starts} \_ \text{with}([1,2,3],X) \).

(b) \( \text{starts} \_ \text{with}(X,[1,2,3]) \).

(c) \( \text{starts} \_ \text{with}([X],[1,2,3]) \).

(d) \( \text{starts} \_ \text{with}([X|Xs],[1,2,3]) \).

(e) \( \text{starts} \_ \text{with}(A,B) \).

3. (10 points) Now consider the `starts_with` rule again, but with cut:

\[
\begin{align*}
\text{starts} \_ \text{with}([ ], _) & :- ! \\
\text{starts} \_ \text{with}([X|Xs],[X|Ys]) & :- \text{ starts} \_ \text{with}(Xs,Ys). \\
\end{align*}
\]

What are all the answers returned for the following goals? (These are the same as for Question 2.) If there are an infinite number, say that, and include at least the first 3 answers.

(a) \( \text{starts} \_ \text{with}([1,2,3],X) \).

(b) \( \text{starts} \_ \text{with}(X,[1,2,3]) \).

(c) \( \text{starts} \_ \text{with}([X],[1,2,3]) \).

(d) \( \text{starts} \_ \text{with}([X|Xs],[1,2,3]) \).

(e) \( \text{starts} \_ \text{with}(A,B) \).
4. (5 points) Which of the following lists represent valid difference lists? For valid difference lists, what list do they represent?

\[1,2,3\] \[1,2\]
\[1,2,3,4,5\] \[1,2,3\]
\[10,20\] \[T\]
\[1,2,3,4,5\] \[
\]

5. (10 points) Write a Prolog rule last that succeeds if the second argument is the last element of the first argument (which must be a list). For example, last([1,2,3],L) should succeed with L=3.

What are the first 3 answers returned for the goal last(Xs,fred)?

6. (6 points) Suppose that we define an absolute value rule in Prolog with the clpr library:

\texttt{myabs}(X,X) :- \{X\geq0\}.
\texttt{myabs}(X,X1) :- \{X<0, \ X1 = -X\}.

What are all the answers returned for the following goals?

(a) \texttt{myabs}(A,10).

(b) \texttt{myabs}(A,-10).

(c) \texttt{myabs}(A,B), \{B=2*A\).
7. (16 points) Consider the following Ruby classes and mixins. (Feel free to tear this page out of the exam and not hand it in, if you don’t have anything on it you want graded and if you want to have it side-by-side when looking at the expressions on the following page.)

class C1
  def test
    "C1 test " + animal
  end
  def animal
    "squid"
  end
end

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

module M2
  def test
    "M2 test " + super
  end
end

class C2 < C1
  include M1
end

class C3 < C1
  include M1, M2
end

class C4 < C2
  def test
    "C4 test " + super
  end
  def animal
    "octopus"
  end
end

Suppose we define the following variables:

c1 = C1.new
c2 = C2.new
c3 = C3.new
c4 = C4.new
What is the result of evaluating the following expressions?

```ruby
c1.test
c2.test
c3.test
c4.test
C4.superclass
C4.ancestors
C4.class
C4.class.class
```

(Hint: `Object.ancestors` evaluates to `[Object, Kernel, BasicObject].`)

8. (15 points) A bag (also known as a multiset) is like a set but can contain duplicate elements. It is unordered. Define a Ruby class `Bag`. It should be a subclass of `Object` and should mix in `Enumerable`. You can use one of the existing Ruby datatypes, such as an array or a hash, to store the contents of the bag. `Bag` should define the following methods:

- `initialize` Initialize this to be an empty bag.
- `add(element)` Add an element to this bag. (This changes the bag.)
- `union(other)` Return a new bag that is the union of the receiver and `other`. This doesn’t change the bag (no side effect).
- `intersect(other)` Return a new bag that is the intersection of the receiver and `other`. This doesn’t change the bag (no side effect).
- `each`Needed for the `Enumerable` mixin.

You can define other helper methods as well; if you do, make them protected.

Examples: Suppose `b1` is a bag containing "squid", "squid", "clam" (in other words, 2 squids and a clam), and `b2` is a bag containing "squid", "squid", "squid" (in other words, 3 squids). Then `b1.union(b2)` returns a bag with 3 squids and a clam, and `b1.intersect(b2)` returns a bag with 2 squids. The total number of squids in `b1.union(b2)` is the maximum of the number of squids in `b1` and the number of squids in `b2`, the total number of squids in `b1.intersect(b2)` is the minimum of the number of squids in `b1` and the number of squids in `b2`, etc. Finally, `b2.each { |x| puts x }` should print squid 3 times.
9. (9 points) Consider the following Java code fragments. In each case, does the code compile correctly? If so, does it execute without error, or is there an exception? Hint: Ellipse2D.Double is a subclass of Ellipse2D, Rectangle2D.Double is a subclass of Rectangle2D, and both Ellipse2D and Rectangle2D are subclasses of RectangularShape.

```java
RectangularShape r = new Rectangle2D.Double(0.0, 0.0, 50.0, 100.0);
Ellipse2D[] a1 = new Ellipse2D[100];
RectangularShape[] a2;
a2 = a1;
a2[0] = r;

RectangularShape r = new Rectangle2D.Double(0.0, 0.0, 50.0, 100.0);
Ellipse2D[] a1 = new Ellipse2D[100];
RectangularShape[] a2;
a2 = a1;
a1[0] = r;

RectangularShape r = new Rectangle2D.Double(0.0, 0.0, 50.0, 100.0);
Rectangle2D[] a1 = new Rectangle2D[100];
RectangularShape[] a2;
a2 = a1;
a2[0] = r;
```

10. (10 points) Write a case for the OCTOPUS eval function to handle or. Your addition should make OCTOPUS handle or exactly as in Racket: it can take 0 or more arguments, and does short-circuit evaluation. Hints: (or #f 2 3) evaluates to 2. Here is the header for the new case:

```octol
eval (OctoList (OctoSymbol "or" : args)) env = .....  
```
11. (6 points) Consider the following definitions in Racket

```
;;; the #:transparent keyword tells Racket to print out the fields of
;;; the struct when you print it
(struct point (x y) #:transparent #:mutable)
```

```
(define (test1 p)
  (set-point-x! p 100)
  (display p))

(define (test2 p)
  (set! p (point 100 200))
  (display p))

(define a (point 0 0))
(define b (point 0 0))
```

(a) What gets printed if we evaluate `(test1 a)`?

What is the value of `a` afterward? (You don’t need to get the syntax exactly right for the way that structs print.)

(b) What gets printed if we evaluate `(test2 b)`?

What is the value of `b` afterward?

12. (10 points) True or false?

   (a) In Java, adding an upcast can never change whether or not a program compiles.

   (b) In Racket, if variables `x` and `y` are aliased, `(eq? x y)` always evaluates to `#t`.

   (c) In Racket, if variables `x` and `y` are aliased, `(equal? x y)` always evaluates to `#t`.

   (d) In Ruby, a class can have at most one superclass, but can have more than one mixin.

   (e) In Ruby, only classes that mix in the `Enumerable` mixin are allowed to implement the `each` method.