CSE 341 - Programming Languages
Final exam - Autumn 2015 - Answer Key

1. (5 points) Which of the following Prolog lists represent valid difference lists? For valid difference lists, what list do they represent?

```
[1,2,3,4]\[1,2,3,4]
valid - represents []

[squid,clam]\[squid]
not valid

[squid,clam]\[clam]
valid - represents [squid]

[X,Y]\Y
not valid

X\X
valid - represents []
```

2. (10 points) Define a Ruby class `Book`. A book should have 3 instance variables: an author (a string), a title (also a string), and an optional year (an integer or nil, representing the year of publication if known). There should be getter methods for author, title, and year (but not setters).

   In addition, we want to be able to sort books, so mix in the `Comparable` mixin. Hint: define the `<=>` method. The expression `a<=>b` should return -1 if `a` comes before `b`, 1 if `b` comes before `a`, and 0 if they are the same. Sort books first by author, then title, then year. For simplicity, assume the authors' names are written last name then first name, so that you can just compare the author strings. Also, a `nil` year is a wild card, so a `nil` year matches any actual year or another `nil`. For example, suppose we have the following books:

```
a = Book.new("Markoff, John", "What the Dormouse Said", 2005)
b = Book.new("Markoff, John", "Machines of Loving Grace", 2015)
c = Book.new("Markoff, John", "What the Dormouse Said", nil)
d = Book.new("Rowling, J.K.", "Harry Potter and the Deathly Hallows", 2007)
```

   Then `a<=>b` evaluates to 1 since the authors are the same and the title of `b` comes before the title of `a`. `a<=>c` evaluates to 0 since the author and title are the same, and the nil year in `c` matches 2005 in `a`. Finally `a<=>d` evaluates to -1 since we first compare the authors.

   Incidentally, John Markoff is speaking in CSE on December 17 — should be interesting! As far as I know J.K. Rowling won't be visiting however.

```
class Book

  attr_reader :author, :title, :year

  def initialize(author, title, year=nil)
    @author = author
    @title = title
    @year = year
  end

  def <=> other
    # compare the authors, using <=> defined on strings
    compare_authors = @author <=> other.author
    # if they are unequal we are done - return that result
    return compare_authors if compare_authors != 0
    # authors are the same - go on to check the titles
    compare_titles = @title <=> other.title
    return compare_titles if compare_titles != 0
    # titles are also the same - check the year
    return 0 if @year==nil || other.year==nil
    @year<=>other.year
  end

end
```

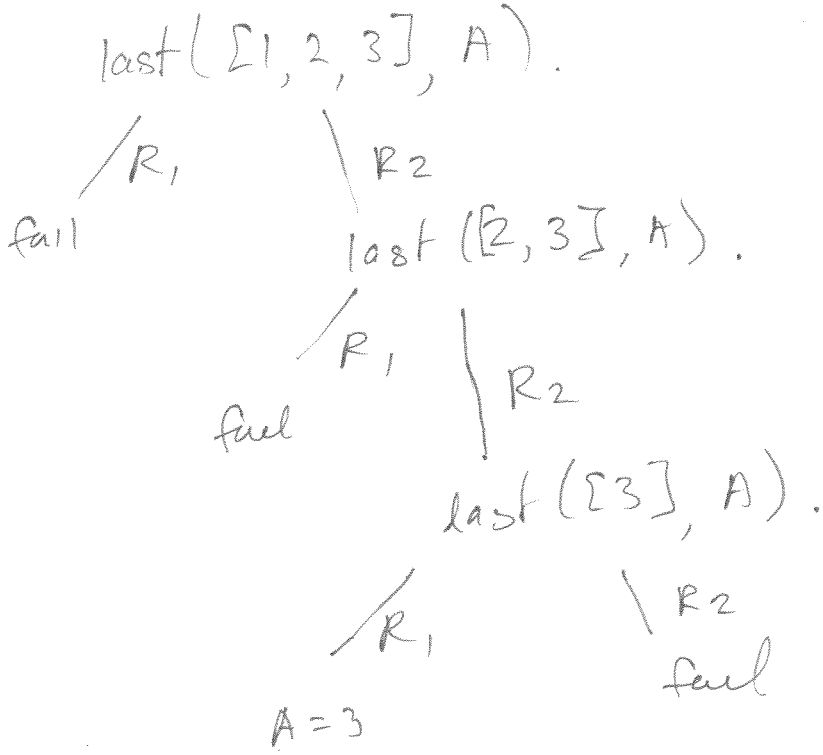
3. (8 points) Consider the following Prolog rule `last`, which succeeds if the first argument is a list, and the second argument is the last element of that list.

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

Draw the search tree for the goal `last([1,2,3],A).`

(R1) last([X], X).

(R2) last([_|Xs], Y) :- last(Xs, Y).



4. (10 points – 8 plus 2 points for clpfd) Write a Prolog rule `take` that succeeds for a goal `take(N,Xs,Ys)` if `Ys` is a list that is `N` elements long and is also the first `N` elements of `Xs`. (This rule is similar to `take` in Haskell except that it should work with variables for different arguments, and also insists that there be enough elements in `Xs` — so `take(3,[a],Ys)` would fail.) For the full 10 points, use the clpfd finite domain constraint solver — otherwise just use ordinary Prolog for 8 points max.

   Here are some examples.

   `take(2,[a,b,c,d,e],X)` succeeds with `X=[a,b]`
   `take(0,[],X)` succeeds with `X=[]`
   `take(2,[],X)` fails
   `take(0,[a,b,c,d,e],X)` succeeds with `X=[]`
   `take(2,Xs,[a,b])` succeeds with `Xs=[a,b|_G01]`, where `_G01` is a name for a fresh variable generated by Prolog

   Here are some additional examples that should succeed if you used clpfd.

   `take(N,Xs,[a,b,c])` succeeds with `N=3, Xs=[a,b,c|_G02]`
   `take(N,Xs,Ys)` succeeds an infinite number of times, first with `N=0, Ys=[]`, then `N=1, Xs=[_G03|G04], Ys=[_G03]`, then `N=2, Xs=[_G05,_G06|G07], Ys=[_G05,_G06]`, and so on.

```
:- use_module(library(clpfd)).

take(0,_,[]).

/* alternate version that insists the second argument be a list:
  take(0,[],[]).
  take(0,[_|_],[]).
*/

take(N,[X|Xs],[X|Ys]) :- N#>0, N1#=N-1, take(N1,Xs,Ys).

/* version in Prolog (does not count for full credit) */
prolog_take(0,_,[]).
prolog_take(N,[X|Xs],[X|Ys]) :- N>0, N1 is N-1, prolog_take(N1,Xs,Ys).
```

5. (10 points) To check whether two strings in Ruby consist of the same characters, the == method for `String` has to compare them character by character — we can't just test whether the references to two strings `s1` and `s2` are identical, since there might be two different objects. However, sometimes we want to be able to compare two string-like objects quickly. For this purpose Ruby has a class `Symbol`, which has the property that if there are two variables referring to symbols with the same characters, there is just one symbol referred to by both variables. This lets us compare two symbols by just comparing their object references. To implement this, when a program creates a new symbol, Ruby checks whether it already exists, and if so returns the already-existing one. Otherwise it creates a new symbol and *interns* it so that it will be found the next time. (Many other languages, Racket for example, provide the same facility.)

   Write a class `MySymbol` that provides the same functionality. So if we evaluate

```
a = MySymbol.new("clam")
b = MySymbol.new("squid")
c = MySymbol.new("squid")
```

   `b` and `c` will refer to the same object (but of course `a` will refer to a different one).

   Hint: to define a class method in `MySymbol`, write it like this:

```
def MySymbol.test(s)
   # method body here
end
```

This class method in `MySymbol` will override a method named `test` in class `Class`, and you can access the inherited method in the usual way, using `super`.

```
class MySymbol

  @@symbols = Hash.new
  attr_reader :str

  def initialize(str)
    @str = str
    @@symbols[str] = self
  end

  def MySymbol.new(str)
    if @@symbols.member?(str)
      return @@symbols[str]
    end
    super
  end

end
```

6. (6 points)

   What does the following Ruby program output?

```
def test(a1,a2,a3,a4)
  a1[0] = 10
  a2[0] = 20
  a3[0] = 30
  a4 = [2,3,4]
end

b1 = [1,2,3]
b3 = [1,2,3]
b4 = [1,2,3]
test(b1,b1,b3,b4)
puts b1.to_s
puts b3.to_s
puts b4.to_s

OUTPUT:
[20, 2, 3]
[30, 2, 3]
[1, 2, 3]
```

7. (6 points) What would the program in Question 6 output if `a1`, `a2`, `a3`, and `a4` were passed by reference?

```
OUTPUT:
[20, 2, 3]
[30, 2, 3]
[2, 3, 4]
```

8. (8 points) Consider the following Racket program.

```
(define s (cons 1 null))

(define (cons-it x)
  (printf "first time: x=~a  s=~a\n" x s)
  (set! s (cons 2 s))
  (printf "second time: x=~a  s=~a\n" x s)
  (set! s (cons 3 s))
  (printf "third time: x=~a  s=~a\n" x s))

(cons-it (cons 100 s))
```

(a) What is the output in normal Racket?

```
first time: x=(100 1)  s=(1)
second time: x=(100 1)  s=(2 1)
third time: x=(100 1)  s=(3 2 1)
```

(b) What would the output be if x were passed by reference?
The same!

```
first time: x=(100 1)  s=(1)
second time: x=(100 1)  s=(2 1)
third time: x=(100 1)  s=(3 2 1)
```

(c) What would the output be if x were passed by name?

```
first time: x=(100 1)  s=(1)
second time: x=(100 2 1)  s=(2 1)
third time: x=(100 3 2 1)  s=(3 2 1)
```

(d) Rewrite the example to simulate call by name by passing a lambda.

```
(define (cons-it-by-name x)
  (printf "first time: x=~a  s=~a\n" (x) s)
  (set! s (cons 2 s))
  (printf "second time: x=~a  s=~a\n" (x) s)
  (set! s (cons 3 s))
  (printf "third time: x=~a  s=~a\n" (x) s))

(cons-it-by-name (lambda () (cons 100 s)))
```

9. (9 points) This question is about the differences between normal Haskell and a version of Haskell that uses call-by-value.

   (a) Are there any functions and function invocations that terminate without error in both normal Haskell and call-by-value Haskell but that return different results? If so give such a function and invocation. (You can use helper functions if necessary in your example.)

   No, there aren't any. (If they work, they give the same answers.)

   (b) Are there any functions and function invocations that terminate in both normal Haskell and call-by-value Haskell, but that give an error in one version of the language but not the other? If so give such a function and invocation. (You can use helper functions if necessary in your example.)

   Yes — for example if the function doesn't evaluate one of its arguments, and we give it an expression for that argument that would give an error if evaluated, the expression works in normal Haskell and we get an error for call-by-value Haskell. (It will always be call-by-value Haskell that has the error.) Example (using `const` and `head` from the Prelude): `const 42 (head [])`

   (c) Are there any functions and function invocations that terminate in one of normal Haskell and call-by-value Haskell, but not the other? If so give such a function and invocation. (You can use helper functions if necessary in your example.)

   Yes — similarly, for example if the function doesn't evaluate one of its arguments, and we give it an expression for that argument that wouldn't terminate if evaluated, the expression works in normal Haskell and we get non-termination for call-by-value Haskell. (It will always be call-by-value Haskell that doesn't terminate.) Example (using `const` and `elem` from the Prelude): `const 42 (0 `elem` [1..])`

10. (9 points) This question is the same as Question 9, except that we are considering normal Haskell and a version of Haskell that uses call-by-name.

   (a) Are there any functions and function invocations that terminate without error in both normal Haskell and call-by-name Haskell but that return different results? If so give such a function and invocation. (You can use helper functions if necessary in your example.)

   No, they always give the same answers. (Call-by-name Haskell might be slower but that's the only difference.)

   (b) Are there any functions and function invocations that terminate in both normal Haskell and call-by-name Haskell, but that give an error in one version of the language but not the other? If so give such a function and invocation. (You can use helper functions if necessary in your example.)

   No.

   (c) Are there any functions and function invocations that terminate in one of normal Haskell and call-by-name Haskell, but not the other? If so give such a function and invocation. (You can use helper functions if necessary in your example.)

   No.

11. (9 points) This question uses the following hierarchy of Java classes for geometric objects.

```
public interface GeometricObject {
}

public interface TwoDObject extends GeometricObject {
}

public class Square implements TwoDObject  {
    ....
}
```

```
public interface ThreeDObject extends GeometricObject {
    public double volume();
}

public class Cone implements ThreeDObject  {
    ....
}

public class Sphere implements ThreeDObject  {
    ....
}

public class Cube implements ThreeDObject  {
    ....
}
```

Suppose we have three different implementations of a `total_volume` method to find the total volume of some 3d objects.

```
public static double total_volume1(ArrayList<ThreeDObject> objects)
{
    double sum = 0;
    for(ThreeDObject obj : objects) {
        sum = sum + obj.volume();
    }
    return sum;
}

public static double total_volume2(
    ArrayList<? extends GeometricObject> objects)
{
    double sum = 0;
    for(GeometricObject obj : objects) {
        sum = sum + ((ThreeDObject) obj).volume();
    }
    return sum;
}

public static double total_volume3(
    ArrayList<? extends ThreeDObject> objects)
{
    double sum = 0;
    for(ThreeDObject obj : objects) {
        sum = sum + obj.volume();
    }
    return sum;
}
```

Now suppose that we have four variables declared as follows:

```
ArrayList<GeometricObject> geo_objects;
ArrayList<ThreeDObject> three_d_objects;
ArrayList<Cube> cubes;
double volume;
```

Consider the following code that uses these methods and variables. Write "compiles" next to the following statements that compile correctly, and write "doesn't compile" next to the ones that don't compile. Finally, if there is a possibility that the statement will cause a cast exception to be raised, write "possible exception" next to it.

`volume = total_volume1(geo_objects);` DOESN'T COMPILE

`volume = total_volume1(three_d_objects);` COMPILES

`volume = total_volume1(cubes);` DOESN'T COMPILE

`volume = total_volume2(geo_objects);` COMPILES; POSSIBLE EXCEPTION

`volume = total_volume2(three_d_objects);` COMPILES

`volume = total_volume2(cubes);` COMPILES

`volume = total_volume3(geo_objects);` DOESN'T COMPILE

`volume = total_volume3(three_d_objects);` COMPILES

`volume = total_volume3(cubes);` COMPILES

12. (10 points) True or false?

    (a) There can be at most one occurrence of a given variable in the pattern for a function definition in Haskell.
    TRUE

    (b) There can be at most one occurrence of a given variable in the head of a rule in Prolog.
    FALSE

    (c) A Ruby class can have at most one mixin.
    FALSE

    (d) Ruby procs capture their environment of definition, but Ruby blocks do not.
    FALSE

    (e) Using the `instance_of?` message in Ruby often indicates that the program isn't following the duck typing philosophy.
    TRUE