Instructions
Answer any four questions. (If you answer more, only the first four will count.) Each question is worth 25 points. Please try to make your answers neat and coherent. Remember, if we can’t read it, it’s wrong. Partial credit will be given, so try to put something down for each question (a blank answer always gets 0 points!).

1. (25 points)
Explain how type-inference works in ML. Illustrate the process by solving for the type of the following function

```ml
fun ugh([]) = []
| ugh([]::t) = ugh(t)
| ugh([a]) = [a]
| ugh([a]::(s::t)) = ugh(([a]@s) :: t)
| ugh((a::(b::t))::r) = (a::(b::t)) :: ugh(r)
```

What does this function do?

2. (25 points)
We have seen throughout the semester that memoization is a useful optimization. We normally add memoization by hand, recoding an existing function. However, with a functional language like ML we can do better—we can automate the memoization process. Write an ML function `memoize(f)` that takes an arbitrary function `f` (of type `'a -> 'b`) and returns a function identical to `f` except that memoization is included. That is, whenever the memoized function is called with an argument `a` it has already seen, the value of `f(a)` previously computed is returned without being recomputed. If the memoized function is called with an argument, `b` it has never seen, `f(b)` is computed, but it is also stored within the function so that the answer can be reused if the function is called again with `b`. 
3. (a) (15 points)
Write Prolog facts and rules that define the relation `append3(L1,L2,L3,L4)`. `append3` represents a 3-way list append. That is, `append3(L1,L2,L3,L4)` is true if lists `L1`, `L2` and `L3` can be appended together to form list `L4`. For example, `append3([1,2],[3],[4,5],[1,2,3,4,5])` and `append3([1],[2],[1,2],[1,2])` are true, while `append3([1],[2],[1],[1,2])` is false.

(b) (5 points)
Explain how Prolog would solve the query `append3([1],[2],[3],L)` using your definition of `append3`.

(c) (5 points)
Would your definition of `append3` work correctly for the query `append3(L1,L2,L3,[1,2,3])`? Why?

4. (25 points)
Assume we have a list `L` of integers. Define a Prolog relation `listify(L,M)` that is true if we can divide `L` into one or more sublists, `M`, so that each sublist contains integers in non-decreasing (sorted) order. That is, if `v1` and `v2` are adjacent in `L` and `v1 ≤ v2` then `v1` and `v2` are adjacent in the same sublist of `M`. However if `v1 > v2` then `v2` ends one sublist and `v2` begins the next sublist in `M`. For example,

```
| ?- listify([3,5,1,8,9,2,1,0], [[3,5],[1,8,9],[2],[1],[0]]). 
  yes

| ?- listify([1,2,3,4,5,6],X). 
  X = [[1,2,3,4,5,6]]

| ?- listify([5,4,3,2,1],[5,4,3,2,1]). 
  no
```

Your solution needs to only handle the case where `L` is bound (known).

5. What do each of the following Python program fragments compute? In each case explain why.

(a) (5 points)
```
L=[1,2,3,4]
for i in [1,2,3]:
    L[-i:]=L[:i]
print L
```

(b) (5 points)
```
def f(a=1,b=2):
    return a+b
print f(f(f())),f(f())
```
6. (a) (15 points)
Interfaces in Java are used to specify constant values and methods implemented by a
number of different classes. If a call takes an interface as a parameter, then any class
that implements the interface may be passed as that parameter. This allows a limited
form of polymorphism. For example, given the declaration

```java
interface Cvt2Bool {
    boolean toBool(Object o);
}
```

any class that implements Cvt2Bool has a method toBool that can be used to convert
a class object into a boolean value. Assume we have a method

```java
public static void printArray(Object[] ar, Cvt2Bool c){ ... }
```

in some class. This method will print an array of class objects, starting at ar[0], and
stopping as soon as c.toBool(ar[i]) is false.

Give an example of a class that implements Cvt2Bool, and show a possible imple-
mentation of printArray.

(b) (10 points)
Interfaces are often simplified in Pizza since Pizza makes parametric polymor-
phism directly available. What changes are needed in Cvt2Bool, your implementa-
tion of Cvt2Bool, and your implementation of printArray to exploit Pizza’s
parametric polymorphism? In what ways are the Pizza definitions an improvement
over what you used in Java?