CIS 500 — Software Foundations
Midterm I

February 17, 2010

Name: ______________________________

Email: ______________________________

	Score
1	
2	
3	
4	
5	
6	
7	
8	
Total	

Instructions

- This is a closed-book exam: you may not use any books or notes.
- You have 80 minutes to answer all of the questions.
- The exam is worth 80 points. However, questions vary significantly in difficulty, and the point value of a given question is not always exactly proportional to its difficulty. Do not spend too much time on any one question.
- Partial credit will be given.
- Good luck!

1. (10 points)

(a) Fill in the definition of the Coq function `insertUnique` below.

The function is intended to be applied to a type `X`, an equality function `eq` for `X`, an element `x` of type `X`, and a list `l` of type `list X`. If if `l` already contains `x`, the returned list should be identical to `l`. On the other hand, if `l` does not already contain `x`, then the result should be a list which is identical to `l` except that it also contains `x` at the very end. For example, `insertUnique nat beq_nat 6 [1,2,4,3]` yields `[1,2,4,3,6]`, while `insertUnique nat beq_nat 4 [1,2,4,3]` yields `[1,2,4,3]`.

```
Fixpoint insertUnique (X : Type) (eq : X → X → bool) (x : X) (l : list X)
            : list X :=
```

(b) Why do we need to pass an equality-testing function `eq` as an argument to `insertUnique` instead of just using = to test for equality?

2. (8 points)

   (a) Briefly explain the use and behavior of the `apply` tactic.

   (b) Briefly explain the use and behavior of the `apply... in...` tactic.

3. (10 points)

Consider the following induction principle:

```
bar_ind
     : forall (X : Type) (P : bar X → Prop),
       P (bar1 X) →
       (forall (x : X) (f : bar X), P f → P (bar2 X x f)) →
       (forall (n : nat) (f : bar X), P f → P (bar3 X n f)) →
       forall (f : bar X), P f
```

Write out the corresponding inductive type definition.

```
Inductive bar ________________________ : _____________________________ :=

  | bar1 : _____________________________________________________

  | bar2 : _____________________________________________________

  | bar3 : _____________________________________________________ .
```

4. (10 points) Suppose we make the following inductive definition:

```
Inductive foo (X : Type) : Type :=
  | foo1 : foo X
  | foo2 : X → foo X
  | foo3 : nat → foo X → foo X.
```

Write out the induction principle that will be generated by Coq.

```
foo_ind :
```

5. (10 points) Define an inductive predicate `all_same X l`, which should be provable exactly when `l` is a list (with elements of type `X`) where all the elements are the same. For example, `all_same nat [1,1,1]` and `all_same nat []` should be provable, while `all_same nat [1,2,1]` and `all_same bool [true,false]` should not be.

```
Inductive all_same (X:Type) : list X → Prop :=
```

6. (8 points) Recall the `appears_in` relation, which expresses that an element `a` appears in a list `l`.

```
Inductive appears_in (X:Type) (a:X) : list X → Prop :=
  | ai_here : forall l, appears_in X a (a::l)
  | ai_later : forall b l, appears_in X a l → appears_in X a (b::l).
```

Complete the definition of the following proof object:

```
Definition appears_example : forall x y : nat, appears_in nat 4 [x,4,y] :=
```

7. (8 points)

Consider the following partial proof:

```
Theorem toil_and_trouble : forall n m,
     double n = double m →
     n = m.
Proof.
  intros n m. induction n as [| n'].
  Case "n = O". simpl. intros eq. destruct m as [| m'].
    SCase "m = O". reflexivity.
    SCase "m = S m'". inversion eq.
  Case "n = S n'". intros eq. destruct m as [| m'].
    SCase "m = O". inversion eq.
    SCase "m = S m'".
      assert (n' = m') as H.
      SSCase "Proof of assertion".
        (* stopped here *)
```

Here is what the "goals" display looks like after Coq has processed this much of the proof:

```
2 subgoals

  SSCase := "Proof of assertion" : String.string
  SCase := "m = S m'" : String.string
  Case := "n = S n'" : String.string
  n' : nat
  m' : nat
  IHn' : double n' = double (S m') → n' = S m'
  eq : double (S n') = double (S m')
  ============================
   n' = m'

subgoal 2 is:
 S n' = S m'
```

This proof attempt is not going to succeed. Briefly explain why and say how it can be fixed. (Do not write the repaired proof in detail—just say briefly what needs to be changed to make it work.)

8. (16 points) Give an informal proof, in English, of the following theorem.

```
Theorem distr_rev : forall X:Type, forall l1 l2 : list X,
   rev (l1 ++ l2) = (rev l2) ++ (rev l1).
```

The definition of `rev` is given in the exam Appendix. You may assume without proof the two lemmas `app_nil_end` and `snoc_with_append`, which are also stated in the Appendix.

Proof:

Appendix

The function `rev` is defined as follows.

```
Fixpoint snoc {X:Type} (l:list X) (x:X) : list X :=
  match l with
  | nil    => [x]
  | h :: t => h :: (snoc t x)
  end.

Fixpoint rev {X:Type} (l:list X) : list X :=
  match l with
  | nil    => nil
  | h :: t => snoc (rev t) h
  end.
```

For question 8 you may assume without proof the following two lemmas.

```
Theorem app_nil_end : forall X (l : list X),
  l ++ [] = l.

Theorem snoc_with_append : forall X : Type,
                           forall l1 l2 : list X,
                           forall x : X,
  snoc (l1 ++ l2) x = l1 ++ (snoc l2 x).
```