CIS 500 — Software Foundations

Midterm I

February 16, 2011

Name: ______________________________

Pennkey: ______________________________

Scores:

1	
2	
3	
4	
5	
6	
7	
Total (78 max)	

1. (10 points) Consider the following `Inductive` definition:

```
Inductive ptree (X:Type) : Type :=
  | c1 : X -> X -> ptree X
  | c2 : ptree X -> ptree X -> ptree X.

Implicit Arguments c1 [[X]].
Implicit Arguments c2 [[X]].
```

For each of the following types, define a function (using `Definition` or `Fixpoint`) with the given type.

(a) `nat -> nat -> ptree nat`

(b) `forall X Y : Type, ptree X -> (X -> Y) -> ptree Y`

2. (8 points) Recall the definition of `\/` from Logic.v:

```
Inductive or (P Q : Prop) : Prop :=
  | or_introl : P -> or P Q
  | or_intror : Q -> or P Q.

Notation "P \/ Q" := (or P Q) : type_scope.
```

Write down a term of type `forall (P Q R:Prop), (P \/ Q -> R) -> Q -> R`.

3. (8 points) Recall the inductively defined proposition `le` from `Logic.v`:

```
Inductive le (n:nat) : nat -> Prop :=
  | le_n : le n n
  | le_S : forall m, (le n m) -> (le n (S m)).
```

(a) What is the type of the `le_n` constructor? (I.e., what is printed if we send Coq the command `Check le_n`?)

(b) Write down a term whose type is

```
forall (n:nat), le 2 n -> le 2 (S (S n)).
```

4. (14 points) Recall that a list `l3` is an "in-order merge" of lists `l1` and `l2` if it contains all the elements of `l1`, in the same order as `l1`, and all the elements of `l2`, in the same order as `l2`, with elements from `l1` and `l2` interleaved in any order. For example, the following lists (among others) are in-order merges of `[1,2,3]` and `[4,5]`:

```
[1,2,3,4,5]
[4,5,1,2,3]
[1,4,2,5,3]
```

Complete the following inductively defined relation in such a way that `merge l1 l2 l3` is provable exactly when `l3` is an in-order merge of `l1` and `l2`.

```
Inductive merge {X:Type} : list X -> list X -> list X -> Prop :=
```

5. (16 points) A list `l1` is a *permutation* of another list `l2` if `l1` and `l2` have exactly the same elements (with each element occurring exactly the same number of times), possibly in different orders. For example, the following lists (among others) are permutations of the list `[1,1,2,3]`:

```
[1,1,2,3]
[2,1,3,1]
[3,2,1,1]
[1,3,2,1]
```

On the other hand, `[1,2,3]` is *not* a permutation of `[1,1,2,3]`, since `1` does not occur twice.

Complete the following inductively defined relation in such a way that `permutation l1 l2` is provable exactly when `l1` is a permutation of `l2`. Feel free to create other inductive definitions besides `permutation` if you find it helpful.

```
Inductive permutation {X:Type} : list X -> list X -> Prop :=
```

6. (8 points) Here is an induction principle for an inductively defined type `myT`.

```
myT_ind :
  forall (X : Type) (P : myT -> Prop),
    (forall x : X, P (c1 x)) ->
    (forall s : myT, P s -> forall t : myT, P t -> P (c2 s t))
    forall t : myT, P t
```

What is the definition of `myT`?

7. (16 points) Recall the definition of `double`:

```
Fixpoint double (n:nat) :=
  match n with
  | O => O
  | S n' => S (S (double n'))
  end.
```

Write an informal proof of this theorem:

Theorem: For any natural numbers `n` and `m`, if `double n = double m`, then `n = m`.