Midterm

Note

This is an open-book exam. You may consult any books, papers, or notes, but discussion is strictly forbidden.

Problems

1. Under what condition are $A[w/x][z/y]$ and $A[z/y][w/x]$ equal? Give an inductive proof of the equality (under that condition). (10%)

2. Prove the following sequents using Gentzen's System LK. You may treat Γ and Δ in a sequent $\Gamma \vdash \Delta$ as *multisets* of formulas (to shorten the proof, as the Exchange rules will no longer be needed).

   (a) $\vdash (A \rightarrow B) \rightarrow ((A \rightarrow \neg B) \rightarrow \neg A)$ (10%)

   (b) $\vdash \forall x(P(x) \wedge Q(x)) \rightarrow \exists y P(y) \wedge \forall z Q(z)$ (10%)

3. Recall that a set Γ of propositions is *inconsistent* if, for any proposition B, the sequent $\Gamma \vdash B$ is provable (using the propositional part of System LK).

   Prove that, if there is some proposition A such that $\Gamma \vdash A$ and $\Gamma \vdash \neg A$ are provable, then Γ is inconsistent. (Hint: think of the Cut rule and the sequents "$\Gamma \vdash \neg\neg A$" and "$\Gamma, \neg\neg A \vdash B$", for an arbitrary proposition B) (10%)

4. Give a pair of pre-condition and post-condition that specifies the correctness requirement of the following code segment for finding the smallest element of an array of numbers. Prove that the code indeed satisfies the requirement.

```
$min, i := A[1], 1;$
while $i < n$ do begin
    $i := i + 1;$
    if $A[i] < min$ then $min := A[i];$
end
```

To handle multiple assignments of the form "$x, y := E, F$", the following axiom may be added to Hoare Logic:

$$\frac{}{\{P[E,F/x,y]\}\ x,y := E,F\ \{P\}}$$

Note: $A[t, u/x, y]$ ($T[t, u/x, y]$) denotes the formula (term) obtained from *simultaneously* substituting term t for free occurrences of x and term u for y in formula A (term T). For example, $(x + 1 > y)[y, x/x, y] = y + 1 > x$. (20%)

5. In a virtual department store, customers purchase products from various stores. The department also allows customers to make gift packages (say, Alice's gift from Bob) so that other customers may choose them.
   Consider the following classes:

```
class Shoe : Product {
  public Shoe (int p, String v);
  public int price ();
  public String vendor ();
  // ...
};

class Clothing : Product {
  public Clothing (int p, String v);
  public int price ();
  public String vendor ();
  // ...
}

Shoe nike (1000, "Nike");
Clothing levis (800, "Levi's");
```

   (a) Use the Composite pattern to design two classes: `Product` as component and `Gift` as composite. Write down their class definitions. (5%)

   (b) Suppose Alice's gift contains `nike` and `levis`. Write down the class definition of `Alicesgift`. (3%)

   (c) Use the Prototype pattern to design the class `Catalog` that contains methods `MakeNike`, `MakeLevis`, `MakeAlicesGift`. Redefine classes `Product`, `Shoe`, `Clothing` if necessary. (5%)

6. Consider the following definitions:

```
class A {
  public void foo () {};
  public void bar () {};
};

class B : A {
  public void foo () ;
  public void bar ();
}

class C {
  public void foo () { a.foo (); };
  public void bar () { a.bar (); };
  private A a;
}
```

(a) Is class `B` a subclass of class `A`? (2%)

(b) Is type `B` a subtype of type `A`? (2%)

(c) Is class `C` a subclass of class `A`? (2%)

(d) Is type `C` a subtype of type `A`? (2%)

(e) If a pattern is defined by using class inheritance (like `B`), we call it
___class pattern ___object pattern (2%)

(f) If a pattern is defined by using forwarding (like `C`), we call it
___class pattern ___object pattern (2%)

7. Suppose we would like to implement class `Collection`:

```
class Collection {
  public void add (Node n);
  public void remove ();
  public Node peek ();
  // ...
```

But there are several randomly access data structures that may be used in the implementation. For instance, we may have linked lists:

```
class LinkedList {
  public void insert (int idx, Node n);
  public void delete (int idx);
  public Node first ();
  public Node last ();
};
```

Or arrays:

```
class Array {
  public void insert (int idx, Node n);
  public void delete (int idx);
  public Node first ();
  public Node last ();
};
```

(a) Apply the Bridge pattern to define the class `RADS` that allows `Collection` to change the underlying randomly access data structure implementations. Redefine the class `Collection` if necessary. (5%)

(b) Apply the Factory method pattern to define and implement the `Collection` subclasses `Stack` and `Queue` by the classes `Array` and `LinkedList` respectively. (4%)

(c) Apply the Abstract Factory pattern to define `Stack` and `Queue` so that the client can choose randomly access data structures while creating `Stack` or `Queue` objects. (6%)