Turn in your homework next Friday in class. Write your solutions very neatly.

1. Write a generic classes and methods in Java 1.5 to manipulate a list structure whose elements have alternating types `S` and `T`. That is, the first element has type `S`, the second has type `T`, the third has type `S`, the fourth has type `T`, and so on. All of your answers should successfully type-check in Java 1.5.

   - Fill in the fields of the class below that describes a list with alternating types:

     ```
     class List<S,T> {
       ...
     }
     ```

   - Add two instance methods to the above class:
     (a) A method `insert` that takes an element of appropriate type, inserts it as the first element in the list, and returns the new list;
     (b) A method `swap` that swaps the first and the third element of the list, and returns the resulting list (you may assume that the original list is guaranteed to have at least three elements);

2. Consider an imperative language similar to IMP that has C-like pointers, as follows. The syntax for expressions is:

   $$e \ ::= \ n \ | \ \mathsf{true} \ | \ \mathsf{false} \ | \ x \ | \ \&x \ | \ *x \ | \ \mathsf{null}$$

   Expressions include integers, booleans, and pointers. Expression $\&x$ represents a pointer that points to x. Expression null denotes a null pointer. The dereference $*x$ requires x to hold non-null pointer; then, $*x$ represents the variable that x points to. The commands are the same as in IMP, except for the assignment:

   $$c \ ::= \ \ldots \ | \ e_1 := e_2$$

   The assignment requires e_1 to be an l-value: either a variable, or a dereference of a non-null pointer. It then updates the variable that e_1 represents with the value of e_2.

   - What are the values in this language?
   - Indicate what evaluation relations you need to define and write all of the evaluation rules for expressions and assignments in this language.

3. Consider the subtyping rule for function types.

   - If we change the rule such that both the arguments and the return values are co-variant, the rule is unsound. Show the modified subtyping rule and write a program that type-checks in the presence of this rule, but yields run-time errors.
   - If the arguments and the return values are both contra-variant, the rule is again unsound. Write a program proving that this rule would not be safe.

   For each question, indicate where you use the modified subtyping rules, and where the run-time error occurs.