Computer Science 131, Spring 2002
Assignment 7: Subtyping and Objects

Out: Wednesday March 6

Due: Wednesday, March 13, in class

This assignment requires no programming, just written responses. To help the graders, your answers should be typed up rather than handwritten.

1. [20%] For the first 4 parts, give the most general conditions on $type_1$ and $type_2$ for the given subtyping judgments to be "safe" (not go wrong at run-time) in an ML-like language with subtyping: $type_1 \preceq type_2$ or $type_1 \succeq type_2$ or $type_1 = type_2$. Give a brief and clear explanation.

   (a) $type_1$ `list` $\preceq$ $type_2$ `list`.

   (b) $type_1$ `foo` $\preceq$ $type_2$ `foo`, defined by `type 'a foo = 'a list * 'a list`.

   (c) $type_1$ `bar` $\preceq$ $type_2$ `bar`, defined by `type 'a bar = 'a -> 'a`

   (d) $type_1$ `baz` $\preceq$ $type_2$ `baz`, defined by `type 'a baz = ('a -> int) -> int`.

2. [20%] If $type_1 \preceq type_2$ then we know from class that neither $type_1$ `ref` nor $type_2$ `ref` is a safe subtype of the other (at least without run-time checks). However, is it safe to assign a value of type $type_1$ to a $type_2$ `ref` or a value of type t_2 to a t_1 `ref`? Again, give a brief explanation of your answer.

3. In all of the following cases, assume that we have a class `Food` and two subclasses (and subtypes) of `Food`, `Vegetable` and `Meat`.

   (a) [20%] The C++ language allows a subclass to *override* a method inherited from the superclass with code having a *more specific* return type. Thus, we could have a class `Animal` having a method

   ```
   virtual Food favoriteFood()
   ```

   and a subclass `Cow` which replaces this with a method having type

   ```
   virtual Vegetable favoriteFood()
   ```

   guaranteeing that the favorite food of any cow will be not just some food, but a vegetable.

Is it always safe to make `Cow*` be a subtype of `Animal*`? That is, it it still safe to provide a (pointer to a) `Cow` object where a (pointer to an) `Animal` object is expected? Here "safe" means that the typechecker rejects code that would try to invoke methods appearing only in a subclass using an object which was created by a superclass (and hence doesn't have such methods).

(b) [20%] The Eiffel language, developed by Bertrand Meyer, is another object-oriented language containing many features aimed at large-scale software development. (For example, one can specify preconditions, postconditions, and invariants assertions that get checked at run-time.)

The Eiffel language allows a subclass to *override* a method inherited from the superclass with code having a *more specific* argument types.

Thus we can specify a class `Animal` with a method (still using C++ syntax, since you probably don't know Eiffel)

```
virtual void eat(Food* f)
```

and then *override* this in a subclass `Cow` with a method having type

```
virtual void eat(Vegetable* v)
```

specifying that it is only correct for cows to eat vegetables, rather than arbitrary code.

Show that if subclasses are subtypes, then one can write Eiffel code that type-checks but at run-time feeds meat to a cow (potentially causing the program to crash).

(c) [20%] If a language used the opposite rule from Eiffel, so that subclasses could replace methods with code allowing less-specific arguments, would this be safe? For example, assume the class `Animal` has a method

```
virtual void eat(Vegetable* f)
```

and then *override* this in a subclass `Omnivore` with a method having type

```
virtual void eat(Food* v)
```