Problems

1. Use-site versus declaration-site variance

Ben Bitdiddle has written some generic Java code to run an iterator, storing the results into some output list:

```
interface Iterator<T> { bool hasNext();
                        T next();       }

interface List<T>     { void add(T);    }

static public <T> void copyTo(Iterator<T> it, List<T> out) {
    while (it.hasNext()) out.add(it.next());
}
```

Alyssa P. Hacker has a `Iterator<Integer>` and would like to copy it into an array of type `List<Number>` (Integer is a subtype of Number). However, she discovers that she cannot do so!

(a) There are a few more general type signatures for `copyTo` which will work with Alyssa's use case, without changing the body of `copyTo` at all. Give **TWO** such signatures.

(b) Sally Scallion suggests that if Java only had *declaration-site variance annotations*, in which we annotate type parameters of interfaces as covariant or contravariant, we would then be able to derive that `Iterator<Integer>` was a subtype of `Iterator<Number>`, and so forth. For the `Iterator` and `List` we have defined above, what are their variances?

(c) The definitions of `Iterator` and `List` in the Java SE 7's standard library have more methods than we have written down here. What declaration-site variance should be assigned to the full class definition of `Iterator` and `List`?

(d) Although declaration-site variance is sufficient to make Ben's example work, there are some cases use-site variance (wildcards) are more expressive than declaration-site variance. Give a brief example of use-site variance which cannot be rewritten using declaration-site variance. Write down any interfaces you use.

(e) Cy D. Fect thumbs his nose at this generics nonsense: "In C++," he grumbles, "You can just do the obvious thing, no fussing about with wildcards or variance annotations."

```
template< class InputIt, class OutputList >
void copyTo(InputIt begin, InputIt end, const OutputList& out ) {
    for (; begin != end; ++begin) {
        out->push_back(*begin);
    }
}
```

The following use of `copyTo` violates the subtyping rules we have studied:

```
class A {};
class B : public A {};

int main() {
    std::vector<B*> v;
    std::vector<A*> v2;
    copyTo(v2.begin(), v2.end(), &v);
    return 0;
}
```

Explain **specifically** how the C++ compiler determines that this usage is illegal. An error message is insufficient; please interpret it in your own words.