Homework 7

Due Thursday, December 7

Reading

1. Read the first nine sections (4.5 pages) of Snyder, "Encapsulation and inheritance in object-oriented programming." This is up through the section called "Subtyping" but not the section called "Attribute Visibility," or anything later in the paper.

2. Fisher and Mitchell, "What is an object-oriented programming language?"

3. Stroustrop, "What is object-oriented programming?"

4. Stroustrop, "An overview of C++."

5. Skim Cook, "Interfaces and specifications for the Smalltalk-80 collection classes." The main point of the paper is contained in Figure 5. Look at Figure 5 first and then work backwards to see what parts of the paper you need to read to understand the figure.

Problems

1. (10 points) Suppose that function f has type $A \to B$ and function g has type $C \to D$. What must be the relationship between types A and C and between types B and D to guarantee that type $C \to D$ is a subtype of type $A \to B$?

2. (10 points) This question asks about the differences between subtyping in Smalltalk and C++.

   (a) In Smalltalk, what conditions would allow class S to be a subtype of class T? In other words, what relationship between S and T would allow every operation on T-objects to be applied to S-objects?

   (b) In C++, what conditions would allow class S to be a subtype of T?

   (c) What do you see as the advantages and disadvantages of each approach?

3. (20 points) This problem is concerned with the relationship between two class interfaces. The first interface is for objects that represent integer sets. The second is for sets with an additional public member function that returns the maximum integer in the set. Since the question asks you to compare Smalltalk and C++, the interfaces are written in a "neutral" ML-like notation.

```
class set     (* sets of integers with insert, membership test  *)
public:
  insert    : int -> unit
  is_member : int -> bool
  print     : unit -> unit
end;

class max_set (* sets with additional function to  return max elt  *)
public:
  insert    : int -> unit
  is_member : int -> bool
  print     : unit -> unit
  max       : unit -> int
end;
```

In either language, it would be possible to define *max_set* as a subclass (derived class) of *set*, or define the two independently. Classes implementing both interfaces would be likely to have some private representation that would not appear in the public interface. Suppose we write a routine *count_ten* which, given a set as an argument, counts how many of the numbers $0, 1, \ldots, 10$ are in the set and returns this number. In computing this sum, we will "send the message" $is\_member(i)$ to the set for each $i \leq 10$.

(a) (7 points) Assume that *max_set* is *not* declared to be a subclass (derived class) of *set* and that *count_ten* works properly for any instance of *set*. In Smalltalk, will *count_ten* work properly for any instance of *max_set*? Why or why not? If this depends on how one or both classes is written, explain how.

(b) (7 points) Again assume that *max_set* is *not* declared to be a subclass (derived class) of *set* and that *count_ten* works properly for any instance of *set*. In C++, will *count_ten* work properly for any instance of *max_set*? Why or why not? If this depends on how one or both classes is written, explain how. Assume that we have used casts in some way to defeat the type checker; do not answer that this will not compile.

(c) (6 points) Suppose instead that *max_set* *is* declared to be a subclass (derived class) of *set* and that *count_ten* works properly for any instance of *set*. Would your answer to (a) or (b) change? How? Why?

4. (10 points) A C++ class may have virtual members that may be redefined in derived classes. However, there is no way to “undefine” a virtual (or non-virtual) member. Suppose we add another kind of member, called *phantom* members, that are treated as virtual, but are only defined in derived classes if an explicit definition is given (i.e., a “phantom” function is not inherited unless its name is listed in the derived class). For example, if we have two classes

```
class A {
...
public:
   phantom void f(){...}
   ...
};
class B : public A {
...
public:
   ...
};
```

then f would appear in the *vtbl* for A objects and, if x is an A object, $x.f()$ would be allowed. However, if f is not declared in B, then f might not need to appear in the *vtbl* for B objects and, if x is a B object, $x.f()$ would not be allowed. Is this consistent with the design of C++, or is there some general property of the language that would be destroyed? If so, explain what this property is.