Item 28. Meaning of Pointer Comparison

In C++, an object can have multiple, valid addresses, and pointer comparison is not a question about addresses. It's a question about object identity.

class Shape { ... };
class Subject { ... };
class ObservedBlob : public Shape, public Subject { ... };

In this hierarchy, ObservedBlob is derived from both Shape and Subject, and (because the derivation is public) there are predefined conversions from an ObservedBlob to either of its base classes.

ObservedBlob *ob = new ObservedBlob;
Shape *s = ob; // predefined conversion
Subject *subj = ob; // predefined conversion

The availability of these conversions means that a pointer to an ObservedBlob may be compared to a pointer to either of its base classes.

if( ob == s ) ...
if( subj == ob ) ...

In this case, both of these conditions will be true even if the addresses contained in ob, s, and subj differ. Consider two possible memory layouts for the ObservedBlob object to which these pointers refer, as shown in Figure 7.

**Figure 7. Two possible layouts for an object under multiple inheritance. Under either layout, the object has multiple addresses.**

Under layout #1, s and subj refer to Shape and Subject subobjects within the complete object that have different addresses from the complete object to which ob refers. Under layout #2, the Shape subobject happens to have the same address as the ObservedBlob complete object, so ob
and $s$ contain the same address.

Under either layout, $ob$, $s$, and $subj$ refer to the same $ObservedBlob$ object, so the compiler must ensure that $ob$ compares equal to both $s$ and $subj$. (We can't compare $s$ with $subj$ because they have no inheritance relationship.) The compiler accomplishes this comparison by adjusting the value of one of the pointers being compared by the appropriate offset. For example, the expression

$$ob == subj$$

may be (loosely) translated as

$$ob ? (ob+delta == subj) : (subj == 0)$$

where $delta$ is the offset of the $Subject$ subobject in an $ObservedBlob$. In other words, $ob$ and $subj$ are equal if they're both null; otherwise, $ob$ is adjusted to refer to its $Subject$ base class subobject and then compared to $subj$.

One important lesson to be drawn from these observations is that we must be careful to avoid losing type information when dealing with pointers and references to objects (and in general). Pointers to $void$ are common culprits:

```cpp
void *v = subj;
if( ob == v ) // not equal!
```

Once we've stripped the address contained in $subj$ of its type information by copying it to a $void *$, the compiler has no choice but to fall back on raw address comparison, and with pointers to class objects that's rarely appropriate.