

# Data Abstraction and Object Orientation

## 9.8 Exercises

- 9.22 Suppose that class D inherits from classes A, B, and C, none of which share any common ancestor. Show how the data members and vtable(s) of D might be laid out in memory. Also show how to convert a reference to a D object into a reference to an A, B, or C object.
- 9.23 Consider the `person_interface` and `list_node_interface` classes described in Example ©9.55. If `student` is derived from `person_interface` and `list_node_interface`, explain what happens in the following method call:

```
student s;  
person *p = &s;  
...  
p.debug_print();
```

You may wish to use a diagram of the representation of a `student` object to illustrate the method lookups that occur and the views that are computed. You may assume an implementation akin to that of Figure ©9.8, without shared inheritance.

- 9.24 Given the inheritance tree of Example ©9.56, show a representation for objects of class `student_prof`. You may want to consult Figures ©9.7, ©9.8, and ©9.9.
- 9.25 Given the memory layout of Figure ©9.7 and the following declarations:

```
student& sr;  
gp_list_node& nr;
```

show the code that must be generated for the assignment

```
nr = sr;
```

(Pitfall: Be sure to consider null pointers.)

- 9.26** Consider the following code, written in a language like Java, with mix-in inheritance:

```
class A {
    private int a;
    ...
}
class B extends A implements Runnable {
    private int b;
    ...
}
...
Runnable r = new B();
```

- (a) Draw a diagram of the implementation of a B object, showing fields (data members) and vtable(s).
  - (b) Show (in pseudo-assembly language) the calling sequence for `r.run()` (caller side only). You may assume that `r` has been loaded into register `r1`, and that `run` is a method defined by the `Runnable` interface. You may use as many registers as you need. You need not preserve `r1`. You may assume that the `this` parameter is to be passed in register `r3`.
- 9.27** Standard C++ provides a “pointer-to-member” mechanism for classes:

```
class C {
public:
    int a;
    int b;
} c;
int C::*pm = &C::a;
    // pm points to member a of an (arbitrary) C object
...
C* p = &c;
p->*pm = 3;    // assign 3 into c.a
```

Pointers to members are also permitted for subroutine members (methods), including virtual methods. How would you implement pointers to virtual methods in the presence of C++-style multiple inheritance?

- 9.28** As an alternative to using ⟨method address, `this` correction⟩ pairs in the vtable entries of a language with multiple inheritance, we could leave the entries as simple pointers, but make them point to code that updates `this` in-line, and then jumps to the beginning of the appropriate method. Show the sequence of instructions executed under this scheme. What factors will influence whether it runs faster or slower than the sequence shown in

Example ©9.53? Which scheme will use less space? (Remember to count both code and data structure size, and consider which instructions must be replicated at every call site.)

Pursuing the replacement of data structures with executable code even further, consider an implementation in which the vtable itself consists of executable code. Show what this code would look like and, again, discuss the implications for time and space overhead.

- 9.29 In Eiffel, shared inheritance is the default rather than the exception. Only renamed features are replicated. As a result, it is not possible to tell when looking at a class whether its members will be inherited replicated or shared by derived classes. Describe a uniform mechanism for looking up members inherited from base classes that will work whether they are replicated *or* shared. (Hint: Consider the use of dope vectors for records containing arrays of dynamic shape, as described in Section 7.4.2. For further details, consult the compiler text of Wilhelm and Maurer [WM95, Sec. 5.3].)
- 9.30 In Figure ©9.9, consider calls to virtual methods declared in A, but called through a B, C, or D object view. We could avoid one level of indirection by appending a copy of the A part of the vtable to the D/B and C parts of the vtable (with suitably adjusted `this` corrections). Give calling sequences for this alternative implementation. In the worst case, how much larger may the vtable be for a class with  $n$  ancestors?
- 9.31 In Ruby, an interface (mix-in) can provide not only method signatures (names and parameter lists), but also method *code*. (It can't provide data members; that would be multiple inheritance.) Would this feature make sense in Java? Explain.
- 9.32 Consider the Smalltalk implementation of Euclid's algorithm, presented at the end of Section ©9.6.1. Trace the messages involved in evaluating `4 gcd: 6`.

