3E will run into an issue with circular dependencies between classes.

First, this type of circular dependency is unresolvable:

```
class A {  B b; };
class B { A a; };
```

This is to be read as "class A contains a data member of type B and class B contains a data member of type A."

The first issue is that the compiler will not know about class B when it is parsing the definition for class A. So it will give up there. That is a somewhat fixable issue, though, with a forward declaration:

```
class B;  // forward declaration
class A {  B b; };
class B { A a; };
```

The forward declaration doesn't fully solve the problem though. When the compiler gets to the definition of class A, the compiler sees that A has a data member of type B. It now will understand that B is a class, but it still does not have enough information. In particular, it needs to know how many bytes to allocate for an instance of A. And that depends on B.

This brings in the bigger issue -- the sizes of A and B are unclear. A contains a B. So how many bytes does B need? Well, B contains an A. So how many bytes does A need? And this recurses forever.

In all, the pattern of:

```
class A {  B b; };
class B { A a; };
```

is not possible.

That said, a tweak to this pattern is achievable. In particular:

```
class B;
class A { B *b; }; // <--- this one is now a pointer
class B { A a; };
```

In this scenario, the forward declaration of B allows the compiler to make sense of the type for the "B *b" in A. Further, there is no issue with size here -- pointers are all 8 bytes. So the compiler can conclude that A has 8 bytes. Similarly, B will have 8 bytes.

You will need this pattern for 3E. Also, note that this doesn't work:

```
class B;
class A { B *b;   void foo() { b->foo(); }; };
class B { A a;  void foo() { ; }; };
```

The compiler is comfortable with allocating 8 bytes for a pointer, but the b->foo() call would trip it up. If you encounter that problem, then make sure you are using the traditional file organization. If you put the definition of A::foo() in A.C and if you "#include <B.h>" in A.C, then it should work fine.