1. How do you identify a pure virtual function?

A pure virtual function is identified with a “= 0” before the closing of the semicolon of declaration:

virtual foo myfoo() const = 0;

1. What happens if you to try to declare a variable of some type that is an ADT?

When the class declaration contains at least one pure virtual function, you can’t create an object of that class. The paradigm here is that classes with pure virtual functions serve as a base class for several derived classes. C++ also allows for pure virtual functions to have a definition; such a need could arise if all the base methods such as myFoo() for the base class are defined, but we would still want to make that class abstract. Thus, we would make the prototype virtual to make the base class abstract:

void myFoo(double x) = 0;

And then we could also provide the definition in an implementation:

void BaseADT::myFoo(double x) { foo = x}

1. In multiple inheritance, what happens if two or more of the base classes have members of the same name?

If a derived class does not redefine a function (virtual or not), the class will use the most recently defined version of the function. If you redefine the function in a derived class, it hides all the base class methods of the same name regardless of argument signatures. It *does not* override the base class declaration with the same function signature. An exception occurs with the “covariance of return type” where the *return type* (not the arguments) of a redefined function using the same signature can vary between different implementations of the method throughout different base classes:

class Section

{

public:

virtual void foo(double x);

…

};

class Economy : public Section

{

public:

virtual void foo();

…

}

Economy route;

route.foo() //this works as it hides the last class definition

route.foo(24.5) //this does not work (assumed throughout that “override” is not considered)