Subject: Re: Difficulty with Class declaration
Posted by cbpporter on Thu, 01 Apr 2010 06:39:59 GMT
View Forum Message <> Reply to Message

brokndodge wrote on Thu, 01 April 2010 01:57
in popups.h
```
#include "main.h"
// struct UDMS;
struct Popups : UDMS { //no difference in results whether
                       //": UDMS" is here or not
//stuff
};
```

and in main.h
```
// #include "popups.h"
// class Popups;
struct UDMS : TopWindow {
  struct lilpop;
  Popups lilpop;

//stuff
};
```

Well you are getting slightly closer. Main.h defines UDMS and Popups inherits form it, so this is ok. But main.h is wrong.

```
struct UDMS : TopWindow {
  struct lilpop;
  Popups lilpop;
```

First you have a "struct lilpop;". Struct used like this is used to forward declare a type, in you case the type will be "lolpop". Then you declare a variable of type Popups and also name as the type the struct from a line ago. But the big problem is that here is you include order:

main.h: Defines UDMS, uses Popups
popups.h: Defines Popups, uses UDMS

This time you have a harmful/uncompilable circular reference. UDMS can not have a Popups member, though it can have a pointer to it.

Quote:
Seems that with c++ I do need to fully understand it. Class member or subclass seems like splitting hairs to me.

No, they are at least in principle profoundly different concepts.
Seeing this:

Quote:
```
class UDMS : TopWindow{}; //main class, creates topwindow, menubar and
                          //TabBar, calls Popups::Login then passes the rest
                          //of the work off to other modules.
                          // I had some difficulty with my initial attempt
                          // to call Sales from UDMS so I put together a
                          // simple popup in an effort to learn how all
                          // of this will work together.

class Popups : UDMS {}; // every class will share
                        // many common popups
class Sales : UDMS, Popups{};
```

I think you are expecting classes to behave differently than in C++. Maybe Perl. Maybe Perls has no traditional inheritance and you simulate it with members like one does in C. In "class UDMS : TopWindow{}" UDMS will inherit everything from TopWindow. You do not need to add a TopWindow member to UDMS.

Also "class Sales : UDMS, Popups{};" is wrong because you are inheriting from UDMS twice: once for UDMS and once for Popups, which already inherits from UDMS. You are using multiple inheritance, a quite tricky and rarely needed feature of C++.

Inheritance is a "is a kind of" relationship. Square is a kind of Shape. You do not want to have a Shape and call members from Square while in Shape. You want a Square.

Membership is a "has a" kind of relationship. Like said above, a Square is a kind of Shape. I does not have a Shape inside (as in adding it as a member; on the implementation side it does have one but it is added by the compiler). It has different stuff inside. It may have two points and a color. If you think that Shape is the common ground and it should have a color, then Shape will have a color, Box will be a kind of Shape and it will not directly have its own color, but it will inherit it from Shape, and it will have two points for the corners which Shape does not have.