Subject: Re: NTL - "deep copy semantics"?
Posted by mr_ped on Thu, 06 Sep 2007 01:05:05 GMT
View Forum Message <> Reply to Message

The memory footprint of class VPoint and House is different, which pretty much breaks any chance of direct "polymorphism" in Array.

Pointers to VPoint can be set to point to House too, and the HouseLayer can typecast the pointers back to House pointers in case it is sure there's House and not VPoint upon that pointer.
(or to do checks of type during pointer casting with RTTI)

If you want to keep the DeepCopy behaviour too (i.e. not just to copy pointer value, but actually to create new instance of VPoint or House, copy the data, and set up the pointer to new instance in the copy of PointLayer), you will need to wrap it up in new class like:

```
class PointContainer {
  VPoint *ptr;

  ...here comes copy constructor which will create new instance of VPoint or House, copy data from old pointer, and set up the new pointer to new copy of PointContainer object...
}
```

This is IMHO very cumbersome in C++ and some overall class design should be rather changed, than trying to do it this way.

I think using pointers will be probably the right answer for this, but I think you should avoid copy behavior, and suffice with pointers copying only, which will point to shared instance of data. These things are prone to memory leaks, so be careful with memory releasing or use smart pointers / Garbage collector.

Still the PointLayer will work only with VPoint pointers in functions, whenever you will want to work with House instance, you will need to cast the pointer, and that should be done probably only by HouseLayer function, i.e. you can either write all those functions twice with minor changes, or use templates...

Anyway, your PointLayer vs HouseLayer class suggest your class design is too much data oriented, not enough abstract probably.

If House is on same level of abstraction than VPoint (and your usage in Layers suggests so), it should share the basic functions interface with with it (functions for layers to manipulate with them).

I.e.

```
class VPoint {
 virtual MoveMe(...);
 virtual HideMe(...);
};
```

```
class House : public VPoint {
 aditional data;
 ..custom version of MoveMe() to handle my additional data;
};

class Tree : public VPoint {
 different additional data;
 ..custom version of MoveMe();
};

class Layer {
 array of VPoint pointers;

 void MoveAllMembers() {
   for each in pointers do
     pointers->MoveMe(); //I don't care if you are House or Tree or VPoint, just move.
 }
}
```

Actually here the House and Tree is on same level of abstraction, VPoint is base class and shouldn't be used directly too much probably. (Unless it feels right, like in Layer way)

If you want to do something specific upon House, you should either have different list of House objects only, or to have a way to identify House pointers within Layer, typecast them back on House pointer, and call special House functions...

All this should be outside of Layer class, which is not supposed to make difference between VPoint and House or Tree.

I think I can't give you better advice without having better idea what you are trying to do.. and even then I will probably not help you too much, I never really designed any serious OOP application, so I lack experience, I'm more a theory guy..
