Subject: Re: Questions about static casting Polymorphic Array Elements, iterator, Ptr and Pte
Posted by navi on Mon, 24 Dec 2012 09:57:56 GMT
View Forum Message <> Reply to Message

Dear Mirek,

First of all, Thank you for the quick response.

mirek wrote on Mon, 24 December 2012 09:50
Event better (more "U++ish") is to use

circle& c = a.Create<circle>();

here.

Quote:
in the above example, is this the correct Syntax & correct way to static casting Polymorphic Array Elements?triangle *m = static_cast<triangle *> (&a[1]);

Yep. Although, IME, if you use the trick above, you in fact seldom need to cast later. I guess that you only need to know the final type to initialize it, rest can be taken care about with virtual methods.

I will use the U++ method Create<type>() when I first create and add element into the array.
I am casting element at a later time depending on the "type" variable which can be access via the base type pointer/referance returned by the Array's index operator, to access the derived part of the object. I am under impression that since the Array is of the base type, I cant access the derived part of the object using the operator[] without casting. As it might return a reference of the base type. does Array return a derived type reference if added using Create<type>()? does Array actually remembers type for individual elements?! if so, then its awesome! I don't need casting what so ever at later time.

P.S. I do not have any virtual method in my objects other then the virtual destructor. which is also empty at the point. I mainly need to retrieve and set various public variables exposed by the different kind of objects. for instance circle only has 3 separate int variable, where triangle has an int array of size 3. some objects might even have other different type of variable like String, Date and etc.

mirek wrote on Mon, 24 December 2012 09:50...anyway, shared smart pointers are generally considered "BIG EVIL", something to avoid....

...Yes. Thing is, the U++ way nicely solves most issues about resource management, so that you never call 'delete' in high-level code, but the solution has somewhat weak spot that you have to be careful about dangling pointers. In some cases, Ptr is a good tool to deal with them...

Understood. pte/ptr is NOT Equal to "shared smart pointers"

Thanks & Regards
Navi

---