Subject: Re: Questions about static casting Polymorphic Array Elements, iterator, Ptr and Pte
Posted by mirek on Mon, 24 Dec 2012 08:50:26 GMT
View Forum Message <> Reply to Message

navi wrote on Sun, 23 December 2012 19:58struct shape{
 int type;
}

struct circle : shape{
 circle(){ type=1; }
 int radius;
 int x, y;
}

struct triangle : shape{
 triangle(){ type=2; }
 int x[3], y[3];
}

struct rectangle : shape{
 rectangle(){ type=3; }
 int x[4],y[4];
}

Array<shape> a;

a.Add(new circle);
a.Add(new triangle);
a.Add(new rectangle);

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.

Quote:
How do I create an iterator and static cast the iterator instead? Is this a better way then the previous method?

Well, U++ way is to prefer index based iteration over iterator. Anyway, if you insist, you can of course use iterator.

```
#include <Core/Core.h>

using namespace Upp;

struct Foo {
 int foo;
};

struct Bar : Foo {
 int bar;
};

CONSOLE_APP_MAIN
{
 Array<Foo> x;
 x.Create<Bar>().bar = 54321;

 for(Array<Foo>::Iterator it = x.Begin(); it != x.End(); ++it)
  DUMP(static_cast<Bar &>(*it).bar);
}
```

Quote:
Are there smart pointers in U++?

Well, there are some historical in 'non-canonical' parts of U++ that are not normally part of U++ releases, anyway, shared smart pointers are generally considered "BIG EVIL", something to avoid.

Quote:
I have seen Ptr and Pte in the Manuel but confuse about what do they actually do? do they only assign null when object is destroyed

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...

Quote:
or do they actually destroy pointing object (i.e. manages the object De-Allocation?) when they go out of scope?

No.

Mirek

---