

Re: C++ Conversion functions for pointers

Source: <http://coding.derkeiler.com/Archive/C/ CPP/alt.comp.lang.learn.c-cpp/2004-04/1409.html>

From: B. v Ingen Schenau (bart_at_ingen.ddns.info)

Date: 04/30/04

Date: Fri, 30 Apr 2004 11:22:31 +0200

Paul wrote:

> *Hi group,*
> *I have worked out a way to return different types by subscript from a*
> *Variant class with the prerequisites that UDT's*
> *a)inherit from a Variant class and..*
> *b)they overload the assignment operator to take an argument of type*
> *Variant&.*

The only flaw is that it does not work, because you are invoking undefined behaviour. :-(

>
> *However I have stumbled upon a problem when I try to convert an*
> *integer pointer to a Variant pointer. I can think other ways to*
> *achieve my goal but I would like to build the Variant class in such a*
> *way that is can implicitly convert basic pointer types such as integer*
> *pointers into a Variant pointer.*

You might want to take a look at boost::Any for an example how a proper variant class can be created.

>
> *Here is some code stripped down to only have conversion for integers:*
> `#include <iostream>`
>
> `class Variant{`
> `public:`
> `Variant(){std::cout<<"Construcing Variant(Default).\n";}`
>
> `/*Conversions for basic types*/`
> `Variant(int x):intData(x){std::cout<<"Construcing Variant(int).\n";}`
> `operator int(){return intData;}`
> `int& operator=(const int& rhs){ intData = rhs ; return intData;}`
>
> `virtual Variant& operator=(Variant&){return *this; }`
> `virtual ~Variant(){std::cout<<"Destructing Variant.\n";}`
> `public:`

```

> /* list of basic types*/
> int intData;
> };
>
>
> /* A UDT must derive from Variant */
> /* Also must overload the assignment operator to take a Variant&*/
> class ObjA:public Variant{
> public:
> ObjA():itsData(0){std::cout<<"Construcing ObjA(Default).\n";}
>
> ObjA& operator=(Variant& rhs){
> if(this == &rhs)
> return *this;
> itsData = static_cast<ObjA&>(rhs).itsData;

```

You might want to use a `dynamic_cast` here. That immediately catches any attempt to assign an object that is not really of type `ObjA`.

```

> return *this;
> }
>
> ObjA& operator=(ObjA& rhs){
> if(this==&rhs)
> return *this;
> itsData = rhs.itsData;
> return *this;
> }
>
> void setData(int x){itsData = x;};
> ~ObjA(){std::cout<<"Destrucing ObjA.\n";}
> private:
> int itsData;
> };
>
>
> template<class DEFAULT_TYPE=Variant>
> class ObjArray{
> public:
> ObjArray():itsSize(0){std::cout<<"Construcing(Default ObjArray).\n";}
> ObjArray(int size):itsSize(size){
> std::cout<<"Construcing ObjArray.\n";
> itsArray = new DEFAULT_TYPE[size];

```

Because `itsArray` always has the type `Variant*`, this will only work if `DEFAULT_TYPE` happens to be `Variant`.

For any other type, you will either get a compile error, or you invoke UB later down the line.

```

> }
> ~ObjArray(){std::cout<<"Destrucing ObjArray.\n"; delete [] itsArray;}

```

alt.comp.lang.learn.c-c++: Re: C++ Conversion functions for pointers

You can not use the delete[] operator if there is a mismatch between the static and dynamic types of its operand.

So, in normal language, you can not use 'delete[] itsArray' if itsArray does not point at an array of objects of type Variant.

This is because otherwise the compiler does not know the size of each object, which makes it impossible to call the destructors for the objects with indices 1 and higher.

The same situation arises when you try to index the array.

<snip>

>

> *AHA TIA*

> *Paul.*

>

Bart v Ingen Schenau

--

a.c.l.l.c-c++ FAQ: <http://www.comeaucomputing.com/learn/faq>

c.l.c FAQ: <http://www.eskimo.com/~scs/C-faq/top.html>

c.l.c++ FAQ: <http://www.parashift.com/c++-faq-lite/>