

Re: null assignment in a template

Source: <http://www.tech-archive.net/Archive/VC/microsoft.public.vc.stl/2007-05/msg00051.html>

- *From:* Ulrich Eckhardt <eckhardt@xxxxxxxxxxxxxxxx>
 - *Date:* Fri, 25 May 2007 09:38:17 +0200
-

Steve Wolf wrote:

Please excuse me if there is a better place for this post. I could not find one in MS's communities. My question is about template programming in C++, not really about STL.

comp.lang.c++.moderated is the place for C++ question in general. The unmoderated counterpart exists, but has a much worse signal to noise ratio.

```
// Initialised supplies an initial value in the variable declaration
// itself NOTE: for non-scalar types (such as floating point types)
// this template won't compile due to the fact that it is illegal to
// declare a template with a non-scalar value
// Use Uninitialised instead, and declare the value in its explicit
// ctor
```

I'm not sure what you mean with floating point types. Indeed, template arguments are restricted to object types, function types, integral values and (I think) pointers. All those need to be compile-time constants.

```
template <typename T, T * zero>
struct Initialised<T*, zero>
{
    Initialised() : m_value(zero) { }
    template <typename U> Initialised(const U* that) : m_value(that) { }
    template <typename U> T* & operator = (const U* that) { if (m_value !=
that) m_value = that; return *this; }

    T* & operator = (const int constant) { return m_value = constant; }

    operator T* & () { return m_value; }
    operator const T* & () const { return m_value; }
    T * m_value; // the plain old data
};
```

Re: null assignment in a template

This looks mightily weird. An assignment operator taking an int and one taking a pointer? Also, what is the value of 'zero' for? Isn't it a plain null pointer? If you want a smart pointer that must be initialised, you could use one that doesn't have a constructor without arguments.

What I really want is to allow myself to declare a plain old pointer and guarantee that its initialized in the constructor or point of declaration, to make it much easier to avoid uninitialized variables issues.

Wait: either you need a plain old pointer or you get something with a constructor. You can't modify the behaviour of a raw pointer to be initialised!