Explicit Constructors

Tuesday, August 31, 2004

A reviewer recently noticed the use of the explicit keyword in some of my sample code and wondered what it was for.

The explicit keyword in C++ is used to declare explicit constructors. Explicit constructors are simply constructors that cannot take part in an implicit conversion. Consider the following example:

```
class Array
{
public:
    Array(size_t count);
    // etc.
};
```

This seems innocent enough until you realize that you can do this:

```
Array array = 123;
```

Well that's ugly but hardly something to get excited about, or is it? This is known as an implicit conversion. The trouble is that it can occur in more insidious places. Consider the following function:

```
void UseArray(const Array& array);
```

Because the compiler will attempt to find an implicit conversion, the following code compiles just find:

```
UseArray(123);
```

Yikes! That's terrible. First there is the problem of code clarity. What does this mean? Secondly, because the implicit conversion involves calling the Array constructor, presumably enough storage is being allocated for a new Array object, not to mention the memory reserved for 123 elements contained in the array. Surely this is not what the programmer intended!?

All of these subtleties can be avoided by making the Array constructor explicit:

```
class Array
{
public:
    explicit Array(size_t size);
    // etc.
};
```

Now a programmer needs to be explicit about her use of the Array constructor.

```
Array array(123);
UseArray(Array(123));
```

A thing of beauty.

So why aren't all constructors explicit? Well in a small number of cases it makes sense to allow an implicit conversion to allow user-defined types to behave like built-in types. A canonical example would probably be a high-precision numeric type:

```
Number number = 123;
```

In this case it makes sense to provide a more natural expression.

And finally, if you have two or more required constructor arguments then it does not make sense to make the constructor explicit since it cannot be used in implicit conversions anyway:

```
class UserName
{
public:
    UserName(const std::wstring& principal,
             const std::wstring& authority);
};
```

Making the UserName constructor explicit would have no effect. You might be tempted to give authority a default value and then allow UserName to take part in an implicit conversion. When considering this, you should always think about whether it could introduce an error by default (/kennykerr/archive/2004/07/09/178435.aspx). Consider what happens if we do this.

```
class UserName
{
public:
    UserName(const std::wstring& principal,
             const std::wstring& authority = std::wstring());
    // etc.
};
```

Now the programmer can write the following dubious code:

```
UserName userName = L"kenny@kennyandkarin.com";
```

What's wrong with that? Well the entire string is passed as the principal and the programmer's entirely innocent code is now incorrect. A better solution is to return to the original UserName class design and then the use of the UserName class is clear:

```
UserName username("kenny",
                  "kennyandkarin.com");
```

Since both arguments are required, it is much harder for the programmer to neglect the one, and setting the authority to an empty string would be easier to spot as an error in a code review. You might even have the UserName constructor throw an ArgumentException if the authority is empty.

---

2 Comments

its too good, i understood it at one glance.

— **mounesh sutar** - Friday, December 15, 2006 7:35:22 AM (/kennykerr/Explicit-Constructors#comment-1307)

I don't know what you are talking about... admittedly, I am a noob
with C++, but please consider revising your comments to explicitly communicate your point (rather than leaving it up to the reader to guess what you mean). Noobs read stuff like yours hoping to gain insight!

For instance: describing Array array = 123; as "ugly" does nothing to clarify what issue -- if any -- you are addressing. Likewise,
describing UseArray(123); as "terrible" is hardly helpful. If you hope to communicate to an audience which does not already know what you are getting at, you must be explicit.

— **dpaddy** - Thursday, February 1, 2007 3:45:02 PM (/kennykerr/Explicit-Constructors#comment-1308)

Comments have been disabled for this content.