# Static variable accessed from different compilation units potentially problematic?

Asked 15 years, 3 months ago Modified 15 years, 3 months ago Viewed 2k times

1

When I started as at my first job as software developer I was assigned to create a system for that allows writing and reading C++ values (or objects) from and to a PDF document. This required a system for mapping types to an id and vice versa. The codebase was huge and there were several 'layers' of code (basic framework layer, tools-layer, view-layers, etc..). The proposed solution was to add a header file to each layer with an enum that contains ids for types defined in the particular layer. To avoid conflicts, each the first value of each enum would start with a offset value (1000, 2000, 3000, ...).

As this system seemed a bit clumsy to me I decided to try something else, and came up with class for representing unique ids:

```
// UniqueId.h
class UniqueId
{
public:
    UniqueId();

    operator int() const;

private:
    int mId;
    static int sCounter;
};

// UniqueId.cpp
int UniqueId::sCounter = 0;

UniqueId::UniqueId()
{
    mId = ++sCounter;
}

UniqueId::operator int() const
{
    return mId;
}
```

Instead of an enum, now simply a list of UniqueId objects could be created without fearing for conflicting ids.

The person reviewing my code (software architect of the company) agreed with this at first. However, a day later he changed his mind and told me this system won't work. I vaguely remember him mentioning that there would be problems resulting from multiple compilation units.

Today, some four years later, I still don't understand what the problem could have been. For as far as I know, the static variable will only be defined in one compilation unit (the one for UniqeId.cpp).

So.. since StackOverflow is so abundant with veteran developers I'd like to take the opportunity to ask for some clarification. Would this indeed have been a problem? Or was my idea ok?

c++

Share Improve this question Follow

asked Sep 9, 2009 at 21:40

StackedCrooked

35.5k ● 46 ● 161 ● 289

## 3 Answers

Sorted by: Highest score (default)

4

Is it important that the "types" be assigned the same IDs between different runs of your program? If not, everything should be fine. If so...

How are different "types" grabbing their IDs? One way might be:

```
class FooType
{
private:
    static UniqueId myId;
...
};
```

This invokes the static initialization order fiasco that Fred Larson linked to. Since initialization order of statics is undefined, you might get different values assigned to myId from build to build, or if you're really unlucky, from run to run.

You might instead create instances of `UniqueId` only when needed, say in the object's constructor or as a method with a static variable. For example:

```
class BarType
{
private:
    const UniqueId &getMyId()
    {
        static UniqueId myId;
        return myId;
    }
    ....
};
```

Now you might have to worry about thread safety (if you're developing a multithreaded program). Aside from that, `UniqueId` values depend greatly on the flow of your program, and will quite likely change from run to run. In one run of the program you may never need to instantiate `BarType`, and so it won't claim an ID. In another run you may need a `BarType` early on; in another you may need a `BarType` later.

Even more sinister is that everything might work right for many builds, until you've forgotten all about this setup. Someone adds a new "type", juggles some of the existing types, or quite possibly makes a completely unrelated change. Then all of the sudden everything breaks for one of the reasons listed above.

Share Improve this answer Follow

answered Sep 10, 2009 at 1:16
Managu
9,039 ● 2 ● 31 ● 36

1 The IDs didn't need to be the same between different runs. Good question though. It might have been an issue in case we ever needed to write the IDs to a file as well... – StackedCrooked Sep 10, 2009 at 10:09

I think the fiasco in this case could lead to worse problems then getting different ID values from one build to another. The initialization of the ID counter could happen after some IDs were already issued, which could lead to duplicate IDs. – Fred Larson Sep 10, 2009 at 20:55

@FredLarson, if the ID counter was initialised later, that would mean that the values issued to the types previously would be undefined, right? I'll be going through the link you've provided, but I'd appreciate if you could clarify this for me. – batbrat Mar 7, 2017 at 8:00

1 @batbrat: Yes. Technically, it would be undefined behavior. In practice, if the value of the ID has no particular meaning and just needs to be unique, it probably wouldn't make any difference other than the potential for duplicates. – Fred Larson Mar 7, 2017 at 14:59

4

He was probably referring to the Static Initialization Order Fiasco.

I don't see a problem from what you've posted here. You'd need dependencies on static variables in other translation units to have the fiasco problem.

Share Improve this answer Follow

answered Sep 9, 2009 at 21:50
Fred Larson
62k ● 18 ● 116 ● 177

1

Your code seems fine to me, and definitly way better than enums with reserved values in all "layers" of the system. Even though there was a problem using statics in this way, there would have been solutions, e.g. GUID. You're idea was fine indeed. Remember, software architect is just a fancy name of a developer with an increase wage - it doesn't mean he's more skilled than you ;)

Share Improve this answer Follow

answered Sep 9, 2009 at 22:38

larsmoa

**12.9k** ● 8 ● 63 ● 86