WIKIBOOKS

More C++ Idioms/Checked delete

Contents

Checked delete

Intent

Increase the safety of the `delete` expression.

Also Known As

Motivation and Sample Code

The C++ Standard allows, in 5.3.5/5, pointers to incomplete class types to be deleted with a delete-expression. When the class has a non-trivial destructor, or a class-specific operator delete, the behavior is undefined. Some compilers issue a warning when an incomplete type is deleted, but unfortunately, not all do, and programmers sometimes ignore or disable warnings.

In the following example, `main.cpp` can see the definition of `Object`. However, `main()` calls `delete_object()`, defined in `deleter.cpp`, which *does not* see the definition of `Object`, but only forward declares it. Calling `delete` on a partially defined type like this is undefined behavior which some compilers do not flag.

```
////////////////////
// File: deleter.hpp
////////////////////
// Declares but does not define Object.
```

```
struct Object;
void delete_object(Object* p);

///////////////////
// File: deleter.cpp
///////////////////
#include "deleter.hpp"

// Deletes an Object without knowing its definition.
void delete_object(Object* p)
{
  delete p;
}

///////////////////
// File: object.hpp
///////////////////
struct Object
{
  // This user-defined destructor won't be called when delete is
  // called on a partially-defined (i.e., predeclared) Object.
  ~Object() {
     // ...
  }
};

///////////////////
// File: main.cpp
///////////////////
#include "deleter.hpp"
#include "object.hpp"

int main() {
  Object* p = new Object;
  delete_object(p);
}
```

Solution and Sample Code

The Checked Delete idiom relies on calls to a function template to delete memory, which fails for declared but undefined types, rather than calls to `delete`.

The following is the implementation of boost::checked_delete, a function template in the Boost Utility library. It forces a compilation error by invoking the `sizeof` operator on the parameterizing type, `T`. If `T` is declared but not defined, `sizeof(T)` will generate a compilation error or return zero, depending upon the compiler. If `sizeof(T)` returns zero, checked_delete triggers a compilation error by declaring an array with -1 elements. The array name is *type_must_be_complete*, which should appear in the error message in that case, helping to explain the mistake.

```
template<class T>
inline void checked_delete(T * x)
{
    typedef char type_must_be_complete[ sizeof(T)? 1: -1 ];
    (void) sizeof(type_must_be_complete);
    delete x;
}

template<class T>
struct checked_deleter : std::unary_function <T *, void>
{
    void operator()(T * x) const
    {
        boost::checked_delete(x);
    }
};
```

NOTE: This same technique can be applied to the array delete operator as well.

WARNING: std::auto_ptr does not use anything equivalent to checked delete. Therefore, instantiating an auto_ptr using an incomplete type may cause undefined behavior in its destructor if, at the point of declaration of the auto_ptr, the template parameter type is not fully defined.

Known Uses

- Boost's checked_delete (http://www.boost.org/libs/utility/checked_delete.html).

Related Idioms

References

Retrieved from "https://en.wikibooks.org/w/index.php?title=More_C%2B%2B_Idioms/Checked_delete&oldid=3743428"