The Courthouse and the Bizarre

Stephan Bergmann

September 2015

More news from the love/hate relationship between the LibreOffice code base and C++11 and beyond

C++ Mon Amour

- LO 4.4: GCC 4.6, MSVC 2012; rely on limited C++11 support
- LO 5.0: GCC 4.6, MSVC 2013
  - Variadic templates
  - Initializer lists
  - Default template arguments for function templates
  - Explicit conversion operators
  - Raw string literals
  - Defaulted and deleted functions
- LO 5.1: GCC 4.7, MSVC 2013
  - Non-static data member initializers
  - Alias templates
  - Delegating constructors

C++ Mon Amour

- If any of that is Greek to you:
  - Scott Meyers: *Effective Modern C++,* O'Reilly, 2014
  - https://github.com/cplusplus/draft

auto

Understand the way auto types are deduced:

```
std::vector<FatClass> v;

for (auto i: v) ...

for (auto & i: v) ...

for (auto const & i: v) ...
```

auto

Understand the way auto types are deduced:

```
std::vector<FatClass> v;

for (FatClass i: v) ...

for (FatClass & i: v) ...

for (FatClass const & i: v) ...
```

Destructor Don’t Throw

What is the behavior of **h(true)** and **h(false)**:

```
void f() { throw 1; }

void g() { f(); }

struct S { ~S() { f(); } };

void h(bool b) {
    try
    {
        if(b) { g(); }
        else  { S s; }
    }
    catch (...) {}
}
```

Destructor Don’t Throw

- “A declaration of a destructor that does not have an exception-specification has the same exception specification as if had been implicitly declared [...].” [class.dtor]/3
- “Given [...] an implicitly-declared special member function, the set of potential exceptions of the implicitly-declared member function [...] consists of all the members from the following sets: [...] if [it] is a destructor, the sets of potential exceptions of the destructor invocations for X’s non-variant non-static data members and for X’s virtual and direct base classes.” [except.spec]/16
- “An [...] implicitly-declared special member function [is] considered to have an implicit exception specification, as follows, where S is the set of potential exceptions of the implicitly-declared member function: [...] the implicit exception specification contains all the types in S.” [except.spec]/17

Destructor Don’t Throw

What is the behavior of **h(true)** and **h(false)**:

```
void f() { throw 1; }

void g() { f(); }

struct S { ~S() nothrow { f(); } };

void h(bool b) {
    try
    {
        if(b) { g(); }
        else  { S s; }
    }
    catch (...) {}
}
```

Destructor Don’t Throw

```
commit a4f97070bdb6172c684ec175c3e6e2a550eb9630
Author: Caolán McNamara <caolanm@redhat.com>
Date:   Fri Jan 23 10:17:37 2015 +0000

    avoid terminate on loading fdo73574-3.docx

    by moving work done in dtor to an explicit method

    Change-Id: I99b3843819ea78f2a91a2784ae0243430dacb3d9

diff --git a/writerfilter/source/ooxml/Handler.hxx b/writerfilter/source/ooxml/Handler.hxx
index 642de8c..2b470a6 100644
--- a/writerfilter/source/ooxml/Handler.hxx
+++ b/writerfilter/source/ooxml/Handler.hxx
@@ -55,7 +55,8 @@ class OOXMLFooterHandler : public Properties
     sal_Int32 mnType;
 public:
     OOXMLFooterHandler(OOXMLFastContextHandler * pContext);
-    virtual ~OOXMLFooterHandler();
+    virtual ~OOXMLFooterHandler() {}
+    void finalize();
     virtual void attribute(Id name, Value & val) SAL_OVERRIDE;
     virtual void sprm(Sprm & sprm) SAL_OVERRIDE;
 };
```

C4521: Multiple Copy Constructors

What can possibly go wrong here:

```
template<typename T> class Wrap {
public:

    template<typename... Arg> Wrap(Arg &&... arg):
        x_(new T(std::forward<Arg>(arg)...)) {}

private:
    std::unique_ptr<T> x_;
};

struct SAL_DLLPUBLIC S { Wrap<X> x_; };
```

C4521: Multiple Copy Constructors

What can possibly go wrong here:

```
template<typename T> class Wrap {
public:

    template<typename... Arg> Wrap(Arg &&... arg):
        x_(new T(std::forward<Arg>(arg)...)) {}

private:
    std::unique_ptr<T> x_;
};

struct SAL_DLLPUBLIC S { Wrap<X> x_; };
```

MSVC: “No idea how to handle T(Wrap<T> const &)”

C4521: Multiple Copy Constructors

Whack-a-Mole 1:

```
template<typename T> class Wrap {
public:

    template<typename... Arg> Wrap(Arg &&... arg):
        x_(new T(std::forward<Arg>(arg)...)) {}

    Wrap(Wrap const &) = delete;

};

Wrap<X> x1, x2;
x2 = x1;
```

C4521: Multiple Copy Constructors

Whack-a-Mole 2:

```
template<typename T> class Wrap {
public:

    template<typename... Arg> Wrap(Arg &&... arg):
        x_(new T(std::forward<Arg>(arg)...)) {}

    Wrap(Wrap const &) = delete;
    Wrap(Wrap &) = delete;
};

Wrap<X> x1, x2;
x2 = x1;
```

C4521: Multiple Copy Constructors

```
commit 5fba3a9a0022c3647d098c00ea63593bd1e78e65
Author: Stephan Bergmann <sbergman@redhat.com>
Date:   Fri Sep 4 11:51:40 2015 +0200

    Prevent perfect forwarding ctor from hijacking copy construction attempts

    ...and hope that this will already be enough to help MSVC 2015 re.
    <http://paste.openstack.org/show/444734/>.

    "This leads to beavior that's intuitive only if you've spent so much time around
    compilers and compiler-writers, you've forgotten what it's like to be human," as
    Meyers so aptly put it.

+private:
+    // Prevent the above perfect forwarding ctor from hijacking (accidental)
+    // attempts at ScopedVclPtrInstance copy construction (where the hijacking
+    // would typically lead to somewhat obscure error messages); both non-const
+    // and const variants are needed here, as the ScopedVclPtr base class has a
+    // const--variant copy ctor, so the implicitly declared copy ctor for
+    // ScopedVclPtrInstance would also be the const variant, so non-const copy
+    // construction attempts would be hijacked by the perfect forwarding ctor;
+    // but if we only declared a non-const variant here, the const variant would
+    // no longer be implicitly declared (as there would already be an explicitly
+    // declared copy ctor), so const copy construction attempts would then be
+    // hijacked by the perfect forwarding ctor:
+    ScopedVclPtrInstance(ScopedVclPtrInstance &) = delete;
+    ScopedVclPtrInstance(ScopedVclPtrInstance const &) = delete;
 };
```

Scoped enum to the Rescue?

Warn or don’t warn:

```
enum class E { E1, E2 };

int f(E e)
{
    switch (e)
    {
    case E::E1: return 1;
    case E::E2: return 2;

    default:    return 0;
    }
}
```

Scoped enum to the Rescue?

Warn or don’t warn:

```
enum class E: int { E1, E2 };

int f(E e)
{
    switch (e)
    {
    case E::E1: return 1;
    case E::E2: return 2;

    default:    return 0;
    }
}
```

Erase Me (Not)

What’s the output:

```
Struct S { int n; ~S() { std::cout << n << '\n'; } };

int main()
{
    std::vector v{{1}, {2}, {3}};
    std::cout << "---\n";
    v.erase(v.begin() + 1);
    std::cout << "---\n";
}
```

Erase Me (Not)

That’s the output:

```
Struct S { int n; ~S() { std::cout << n << '\n'; } };

int main()
{
    std::vector v{{1}, {2}, {3}};
    std::cout << "---\n";
    v.erase(v.begin() + 1);
    std::cout << "---\n";
}
```

```
3
2
1
---
3
---
1
3
```

Erase Me (Not)

And that's why:

```
Struct S
{
    int n;
    ~S() { std::cout << n << '\n'; }

    S(S const &) = default;
    S & operator(S const &) = default;
    S & operator =(S && s) {
        std::cout << "move " << s.n " over " << n << '\n';
        n = s.n; return *this;
    }
    S(S &&) = default;
};
```

Erase Me (Not)

And that’s why (or not):

```
int main()
{
    std::vector v{{1}, {2}, {3}};
    std::cout << "---\n";
    v.erase(v.begin() + 1);
    std::cout << "---\n";
}
```

```
3
2
1
---
move 3 over 2
3
---
1
3
```

It’s Never Too Late

```
commit 01d0f0f6adf7e3a340dd44690db5d7d7ed08d3e5
Author: Stephan Bergmann <sbergman@redhat.com>
Date:   Thu Sep 17 14:31:07 2015 +0200

    "unnamed namespaces don't work well yet" is no longer true

    Change-Id: I7a04c2d04e3fc52982d83119755e0b349d232a47

diff --git a/tools/source/fsys/urlobj.cxx b/tools/source/fsys/urlobj.cxx
index 65f451c..51887b7 100644
--- a/tools/source/fsys/urlobj.cxx
+++ b/tools/source/fsys/urlobj.cxx
@@ -46,9 +46,6 @@
 #include <sax/tools/converter.hxx>
 #include <rtl/uri.hxx>

-namespace unnamed_tools_urlobj {} using namespace unnamed_tools_urlobj;
-    // unnamed namespaces don't work well yet...
-
 using namespace css;

 //  INetURLObject
@@ -420,7 +417,7 @@ inline void INetURLObject::appendEscape(OUStringBuffer & rTheText,
     rTheText.append( (sal_Unicode)INetMIME::getHexDigit(int(nOctet & 15)) );
 }

-namespace unnamed_tools_urlobj {
+namespace {

 enum
 {
```

“And all of a sudden I’m relatively sane
With everything to lose and nothing to gain
Or something like that” —Robert Pollard