redhat.

Plug Yourself In

Stephan Bergmann

September 2015

Learn how to write a Clang compiler plugin.
Bring your computer.

A Motivating Example

Spot the bug:

```
PositionHolder::~PositionHolder() try
{
    mxSeekable->seek(mnPosition);
}
catch (...)
{
}
```

A Motivating Example

Right, an empty handler in a destructor's function-try-block just re-throws

```
PositionHolder::~PositionHolder()
{
    try
    {
        mxSeekable->seek(mnPosition);
    }
    catch (...)
    {
    }
}
```

But how to grep for such mistakes?

The Compiler to the Rescue

- Write a Clang plugin that detects function-try-blocks on destructors
  - For simplicity, just find *all* destructor function-try-blocks, not just those with empty handlers. They are all bad, anyway
- Emit warnings/errors at build time
  - For those cases that the build actually sees

Clang Setup

- Need any random Clang version:
  - dnf install clang
- Need the corresponding Clang/LLVM include files:
  - dnf install clang-devel llvm-devel
- Tell LO's autogen.input:
  - --enable-compiler-plugins
  - CC=clang
  - CXX=clang++

- If you build your own Clang:
  - cmake -DLLVM_ENABLE_ASSERTIONS=ON ...
  - CLANGDIR=...

Plugin Basics

compilerplugins/clang/dtortryblock.cxx:

```
#include "plugin.hxx"
namespace {
class DtorTryBlock:
    public clang::RecursiveASTVisitor<DtorTryBlock>,
    public loplugin::Plugin
{
public:
    explicit DtorTryBlock(InstantiationData const & data): Plugin(data) {}
    void run() override {
        if (compiler.getLangOpts().CPlusPlus) {
           TraverseDecl(compiler.getASTContext().getTranslationUnitDecl());
        }
    }
};
loplugin::Plugin::Registration<DtorTryBlock> X("dtortryblock");
}
```

RecursiveASTVisitor

- A (non-virtual) member function to visit each individual kind of node:
  - bool VisitCallExpr(CallExpr * expr);
  - bool VisitFunctionDecl(FunctionDecl * decl);
  - bool VisitIfStmt(IfStmt * stmt);
- See the clang/AST/ include files for the various kinds of nodes:
  - clang/AST/Expr.h, clang/AST/ExprCXX.h
  - clang/AST/Decl.h, clang/AST/DeclCXX.h
  - clang/AST/Stmt.h, clang/AST/StmtCXX.h

But Whom to Visit?

```
$ cat test.cc
struct S { ~S() try {} catch (...) {} };

$ clang++ -fsyntax-only -Xclang -ast-dump test.cc
TranslationUnitDecl 0x5c1bca0 <<invalid sloc>> <invalid sloc>
|-TypedefDecl 0x5c1c558 <<invalid sloc>> <invalid sloc>
        implicit __builtin_va_list 'struct __va_list_tag [1]'
`-CXXRecordDecl 0x5c1c5a8 <test.cc:1:1, col:39> col:8
        struct S definition
  |-CXXRecordDecl 0x5c1c6c0 <col:1, col:8> col:8
        implicit referenced struct S
  `-CXXDestructorDecl 0x5c1c7d0 <col:12, col:37> col:12 ~S 'void (void)'
    `-CXXTryStmt 0x5c1c8f0 <col:17, col:37>
      |-CompoundStmt 0x5c1c8a8 <col:21, col:22>
      `-CXXCatchStmt 0x5c1c8d8 <col:24, col:37>
        |-<<<NULL>>>
        `-CompoundStmt 0x5c1c8c0 <col:36, col:37>
```

Plugin, Refined

```
class DtorTryBlock:
{
public:

    bool VisitCXXDestructorDecl(CXXDestructorDecl const * decl)
    {
        if (ignoreLocation(decl)) {
            return true;
        }
        // ...
        return true;
    }

};
```

Does it Work at All?

```
class DtorTryBlock:
{
public:

    bool VisitCXXDestructorDecl(CXXDestructorDecl const * decl)
    {
        if (ignoreLocation(decl)) {
            return true;
        }
        report(
            DiagnosticsEngine::Warning, "destructor found",
            decl->getLocation())
            << decl->getSourceRange();
        return true;
    }

};
```

Catch the Try-Block

```
bool VisitCXXDestructorDecl(CXXDestructorDecl const * decl) {
    if (ignoreLocation(decl)
        || !decl->doesThisDeclarationHaveABody())
    {
        return true;
    }
    auto s = dyn_cast<CXXTryStmt>(decl->getBody());
    if (s != nullptr) {
        report(
            DiagnosticsEngine::Warning,
            "destructor with function-try-block",
            s->getLocStart())
            << decl->getSourceRange();
    }
    return true;
}
```

Try it out

- git revert bb71e3a40067e4ef6c6879e6d26ad20f728dd822
- make writerperfect

“//TODO” –anonymous