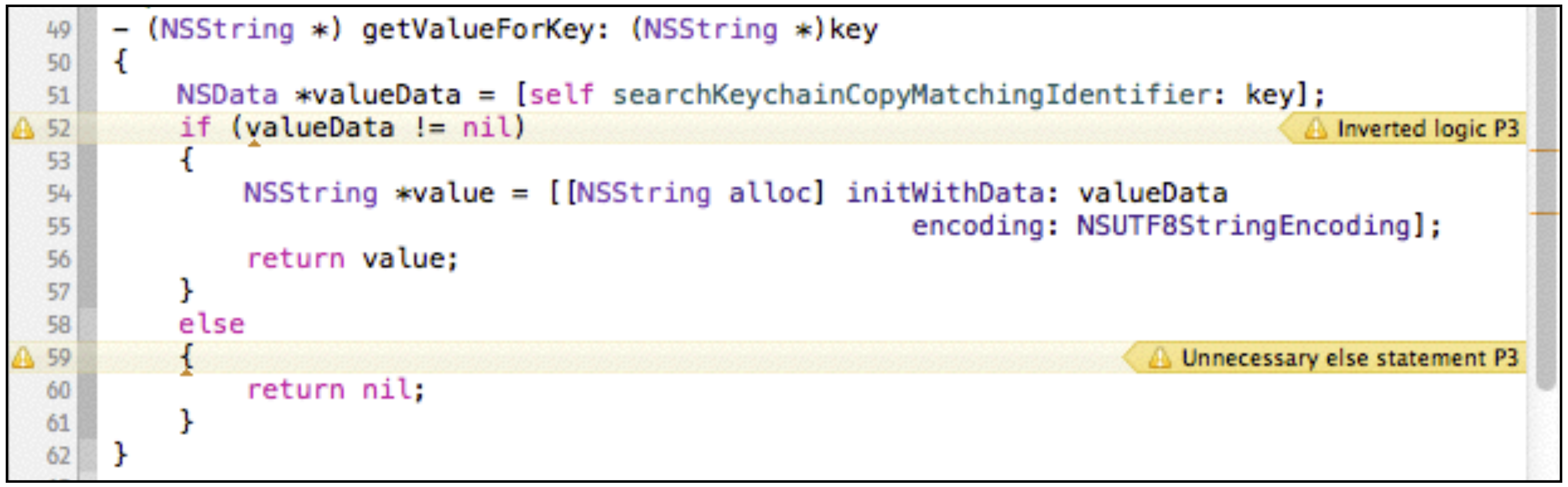
OCLint



```
- (NSString *) getValueForKey: (NSString *)key
{
    NSData *valueData = [self searchKeychainCopyMatchingIdentifier: key];
    if (valueData != nil)                                          Inverted logic P3
    {
        NSString *value = [[NSString alloc] initWithData: valueData
                                                encoding: NSUTF8StringEncoding];
        return value;
    }
    else
    {                                                   Unnecessary else statement P3
        return nil;
    }
}
```

What is it?

OCLint is a static code analysis tool for improving quality and reducing defects by inspecting C, C++ and Objective-C code and looking for potential problems.

Built by Longyi Qi in Texas

Why Should I Use It?

A.K.A. Don't I have enough warnings to squash?

OCLint can catch:

- **Possible bugs** - empty if/else/try/catch/finally statements
- **Unused code** - unused local variables and parameters
- **Complicated code** - high cyclomatic complexity, NPath complexity and high NCSS
- **Redundant code** - redundant if statement and useless parentheses
- **Code smells** - long method and long parameter list
- **Bad practices** - inverted logic and parameter reassignment

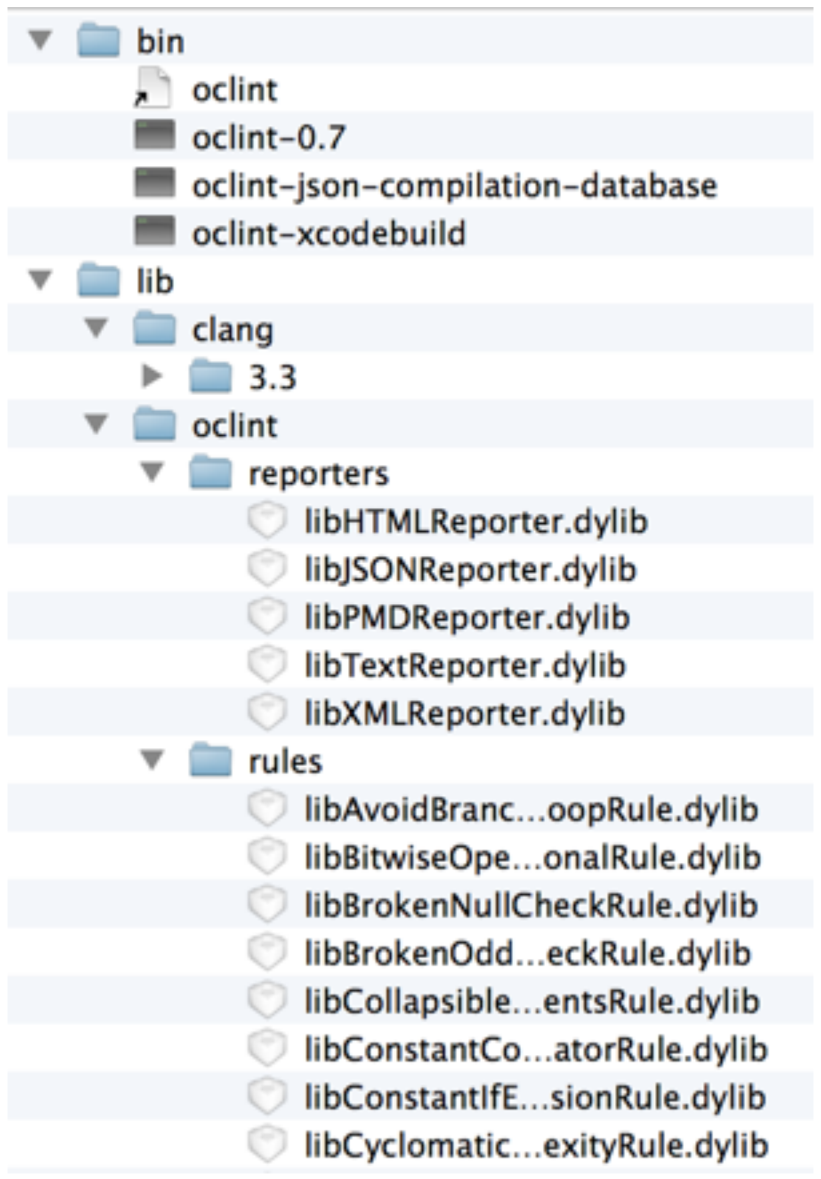
Clang

- Compiler front-end to LLVM
- Built by Chris Lattner at University of Illinois, now at Apple
- Xcode 3.1 actually had GCC as an LLVM front-end, replaced by Clang in 3.2
- A major goal of LLVM is to decouple the front-end compiler from tons of different target hardware configurations
- A way to help decouple that is through the AST

Clang AST

- Super Complicated & Intense
- Abstract Syntax Tree
- 100k LOC alone
- Keeps track of nodes representing everything from methods, to variables and everything in between

Installation



```
▼ bin
    oclint
    oclint-0.7
    oclint-json-compilation-database
    oclint-xcodebuild
▼ lib
  ▼ clang
    ▶ 3.3
  ▼ oclint
    ▼ reporters
        libHTMLReporter.dylib
        libJSONReporter.dylib
        libPMDReporter.dylib
        libTextReporter.dylib
        libXMLReporter.dylib
    ▼ rules
        libAvoidBranc...oopRule.dylib
        libBitwiseOpe...onalRule.dylib
        libBrokenNullCheckRule.dylib
        libBrokenOdd...eckRule.dylib
        libCollapsible...entsRule.dylib
        libConstantCo...atorRule.dylib
        libConstantIfE...sionRule.dylib
        libCyclomatic...exityRule.dylib
```

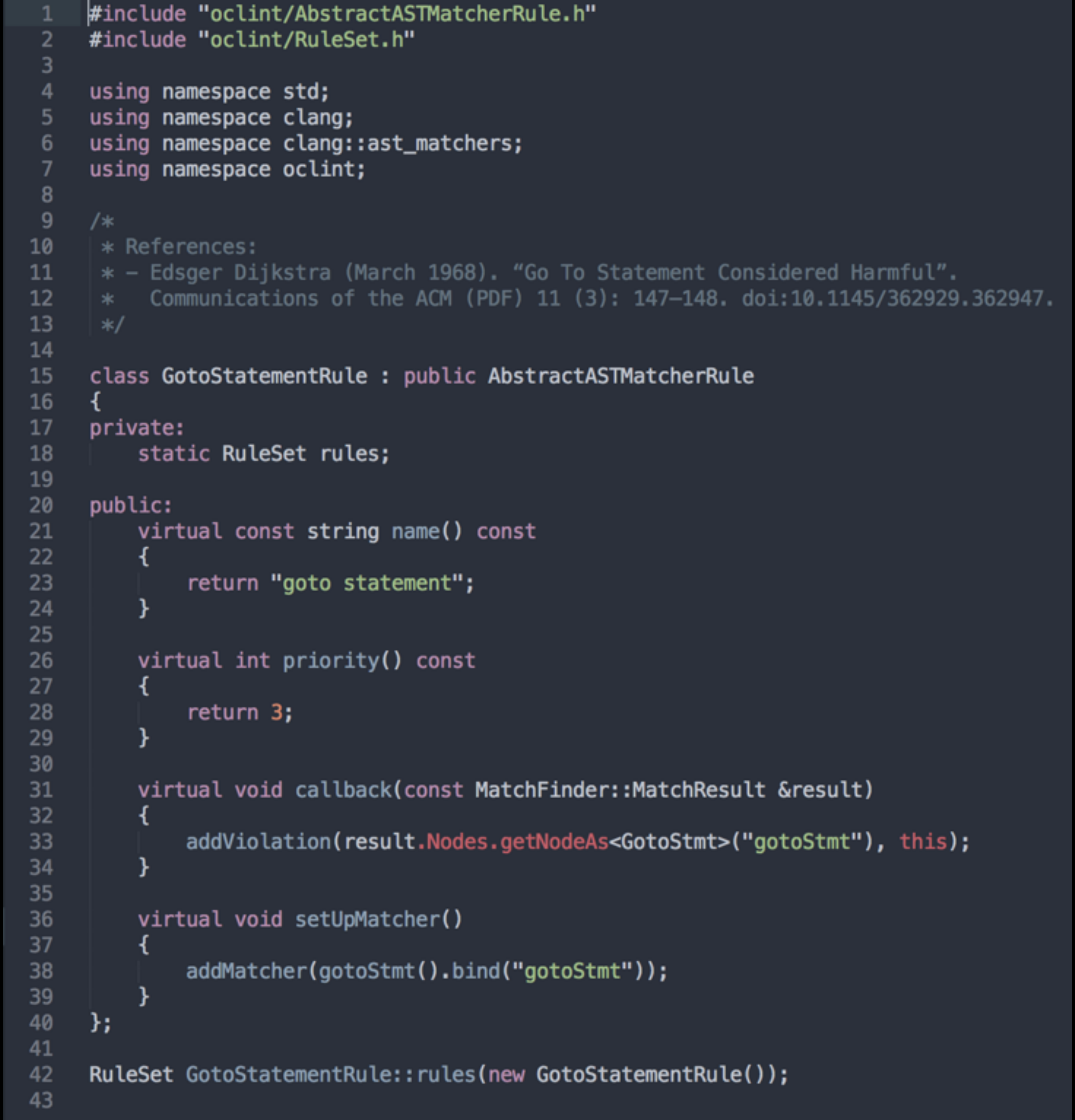
- Pre-compiled binaries
- Build from source (./make)
- Building from source takes about 30m

How are rules made?

- Compiled into dylib's to speed development
- Rule Types
  - AbstractASTMatcherRule
  - AbstractASTVisitorRule
  - AbstractSourceCodeReaderRule

AbstractASTMatcherRule

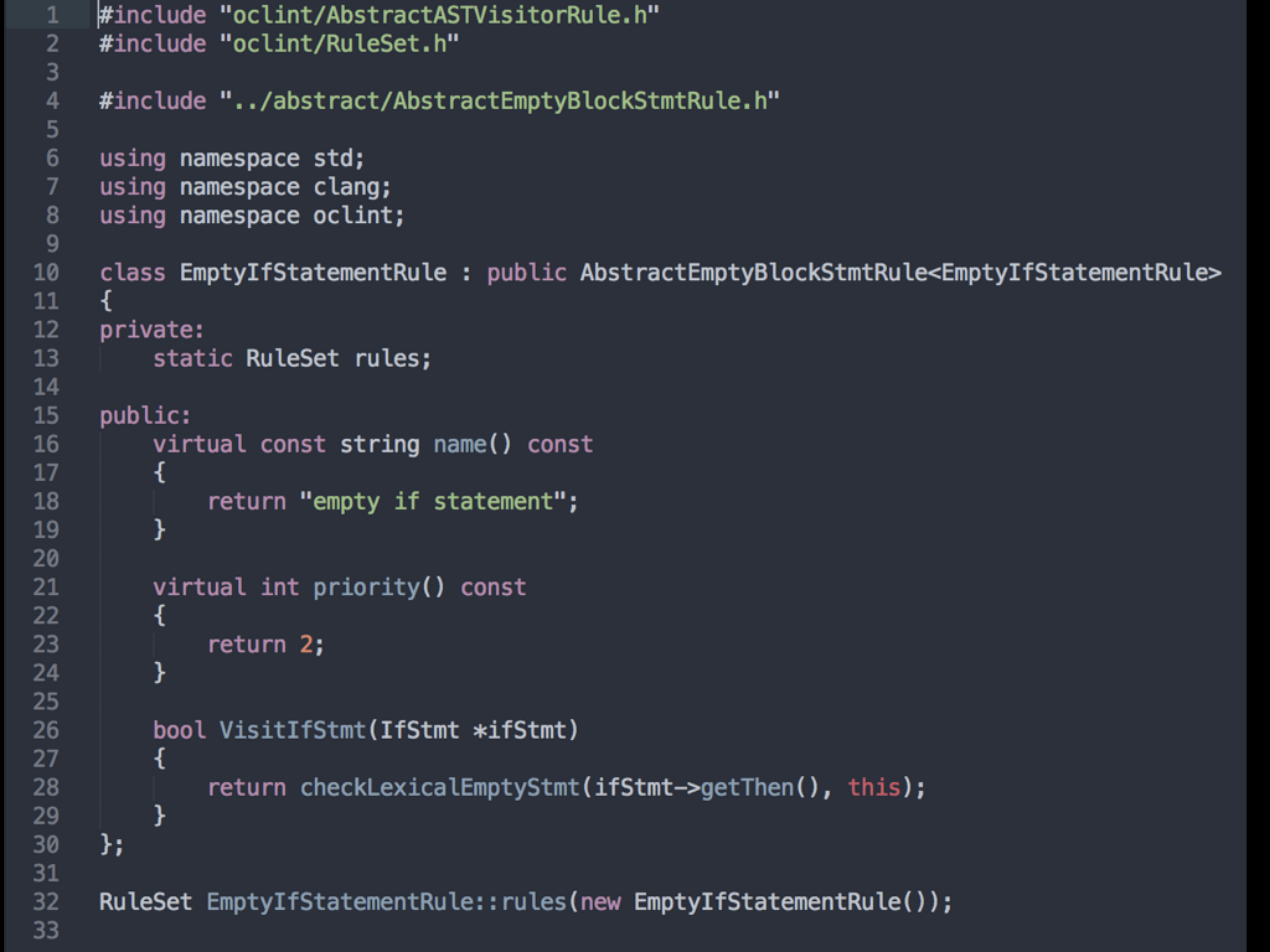
- Looks for exact node in AST
- Ex. “goto statement”
- Queries AST so it’s super fast
- Very few rules are written like this



```cpp
#include "oclint/AbstractASTMatcherRule.h"
#include "oclint/RuleSet.h"

using namespace std;
using namespace clang;
using namespace clang::ast_matchers;
using namespace oclint;

/*
 * References:
 * - Edsger Dijkstra (March 1968). “Go To Statement Considered Harmful”.
 *   Communications of the ACM (PDF) 11 (3): 147–148. doi:10.1145/362929.362947.
 */

class GotoStatementRule : public AbstractASTMatcherRule
{
private:
    static RuleSet rules;

public:
    virtual const string name() const
    {
        return "goto statement";
    }

    virtual int priority() const
    {
        return 3;
    }

    virtual void callback(const MatchFinder::MatchResult &result)
    {
        addViolation(result.Nodes.getNodeAs<GotoStmt>("gotoStmt"), this);
    }

    virtual void setUpMatcher()
    {
        addMatcher(gotoStmt().bind("gotoStmt"));
    }
};

RuleSet GotoStatementRule::rules(new GotoStatementRule());
```

AbstractASTVisitorRule

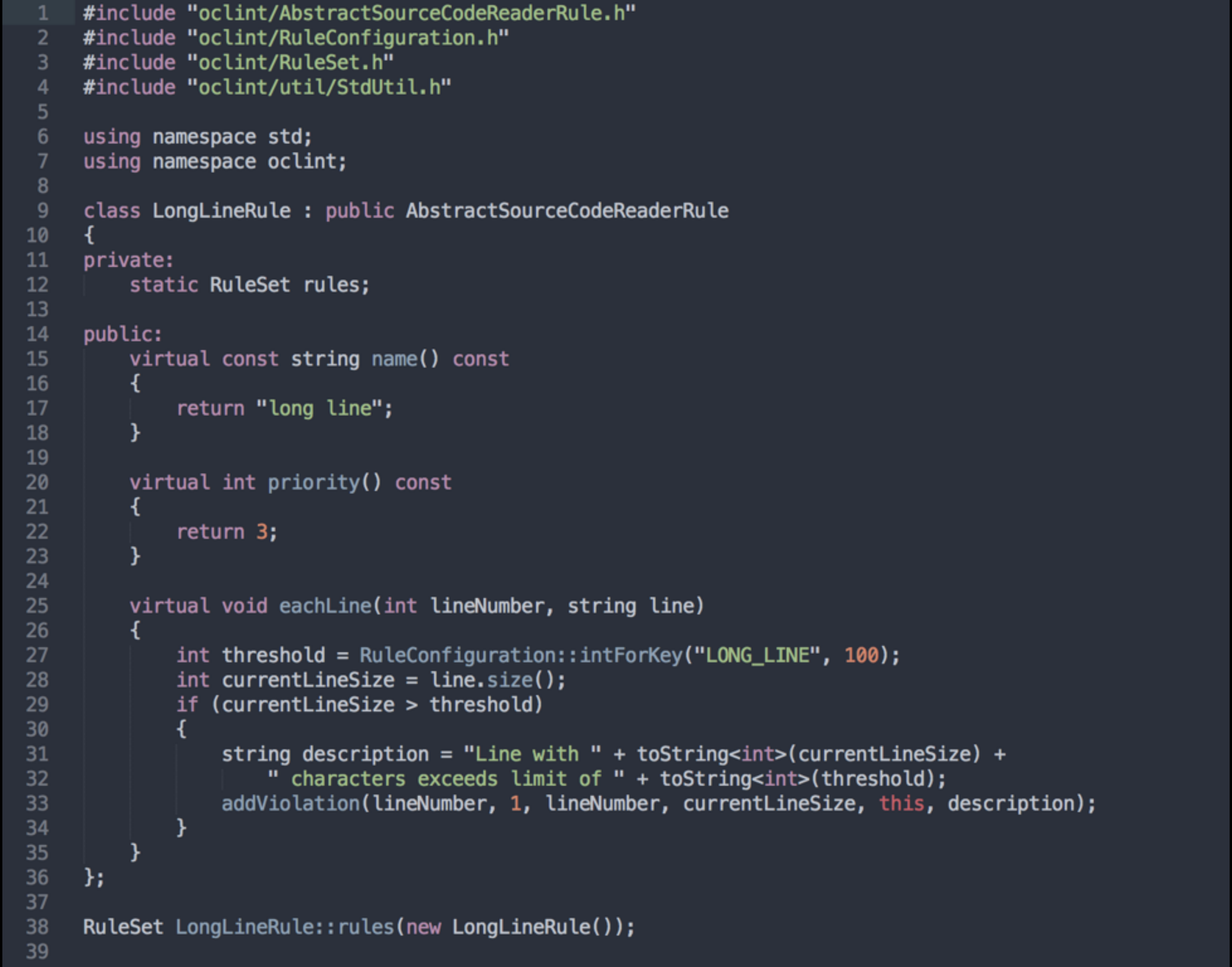
- Looks for specific pattern, then dives down & around
- Most rules are built with this type
- Ex. “empty if statement” or “must call super”
- Parses AST, pretty fast, not as fast as matcher



```cpp
#include "oclint/AbstractASTVisitorRule.h"
#include "oclint/RuleSet.h"

#include "../abstract/AbstractEmptyBlockStmtRule.h"

using namespace std;
using namespace clang;
using namespace oclint;

class EmptyIfStatementRule : public AbstractEmptyBlockStmtRule<EmptyIfStatementRule>
{
private:
    static RuleSet rules;

public:
    virtual const string name() const
    {
        return "empty if statement";
    }

    virtual int priority() const
    {
        return 2;
    }

    bool VisitIfStmt(IfStmt *ifStmt)
    {
        return checkLexicalEmptyStmt(ifStmt->getThen(), this);
    }
};

RuleSet EmptyIfStatementRule::rules(new EmptyIfStatementRule());
```

AbstractSourceCodeReader Rule

- Parses raw text
- Doesn’t use clang
- Only one rule written with this: “line length”
- Obviously slow as molasses



```cpp
#include "oclint/AbstractSourceCodeReaderRule.h"
#include "oclint/RuleConfiguration.h"
#include "oclint/RuleSet.h"
#include "oclint/util/StdUtil.h"

using namespace std;
using namespace oclint;

class LongLineRule : public AbstractSourceCodeReaderRule
{
private:
    static RuleSet rules;

public:
    virtual const string name() const
    {
        return "long line";
    }

    virtual int priority() const
    {
        return 3;
    }

    virtual void eachLine(int lineNumber, string line)
    {
        int threshold = RuleConfiguration::intForKey("LONG_LINE", 100);
        int currentLineSize = line.size();
        if (currentLineSize > threshold)
        {
            string description = "Line with " + toString<int>(currentLineSize) +
                " characters exceeds limit of " + toString<int>(threshold);
            addViolation(lineNumber, 1, lineNumber, currentLineSize, this, description);
        }
    }
};

RuleSet LongLineRule::rules(new LongLineRule());
```

OCLint Command Line

- Works with both xcodebuild & xctool
- You should definitely use xctool (Facebook)
- `xctool -project -scheme -reporter json-compilation-database build`
- Will generate `compile_commands.json`
- then run `oclint-json-compilation-database`

OCLint & Xcode

DEMO TIME!

Resources

- oclint.org
- github.com/oclint/oclint
- clang.llvm.org
- github.com/facebook/xctool
- github.com/travisjeffery/ClangFormat-Xcode