Status report on custom static-code analysis with clang-tidy

Sandro Wenzel

ALICE OFFLINE WEEK; 30.03.2017
The mission

❖ Provide automatic checks (and corrections) of our custom coding conventions + standard C++11 practices

❖ Provide other tools to analyse, inspect, modify source code
Clang-tidy

- Clang-tidy is an extensible code checking framework based on the llvm/clang-libraries
  - does all the heavy lifting (parsing C++; provides convenient API access/callbacks to abstract syntax tree; clang diagnostic compiler messages …)
- large industry community behind
- already implements wide range of checks
- very easily extensible (apart from one drawback)
- integrated ability to autocorrect/fix errors in place

http://clang.llvm.org/extra/clang-tidy/
Example

```c
struct A {
    virtual void foo(int) = 0;
};

struct B : public A {
    virtual void foo(int x) {
        if (x==1)
            printf("hello");
    }
};
```
```cpp
struct A {
    virtual void foo(int) = 0;
};

struct B : public A {
    virtual void foo(int x) {
        if (x==1)
            printf("hello");
    }
};
```

```
clang-tidy  -checks=--*,moder*over*,read*braces* test.cxx -- -std=c++11
/Users/swenzel/test.cxx:8:16: warning: prefer using 'override' or (rarely) 'final' instead of 'virtual' [modernize-use-override]
    virtual void foo(int x) {
        override
/Users/swenzel/test.cxx:9:13: warning: statement should be inside braces [readability-braces-around-statements]
        if(x==1)
```
Example

```cpp
struct A {
    virtual void foo(int) = 0;
};

struct B : public A {
    virtual void foo(int x) {
        if (x==1)
            printf("hello");
    }
};
```

With the automatic fix:

```cpp
struct A {
    virtual void foo(int) = 0;
};

struct B : public A {
    void foo(int x) override {
        if (x==1) {
            printf("hello");
        }
    }
};
```

```
clang-tidy  -checks=-*,moder*over*,read*braces* test.cxx -- -std=c++11
```

```
/Users/swenzel/test.cxx:8:16: warning: prefer using 'override' or (rarely) 'final' instead of 'virtual' [modernize-use-override]
    virtual void foo(int x) {
        ^
    override
/Users/swenzel/test.cxx:9:13: warning: statement should be inside braces [readability-braces-around-statements]
        if(x==1)
```
Adding a custom check

A custom check is a class extending from ClangTidyCheck

```cpp
/// Checking the class member naming convention
class MemberNamesCheck : public ClangTidyCheck {

public:

    void registerMatchers(MatchFinder *Finder) override;

    void check(const MatchFinder::MatchResult &Result) override;

};
```
A clang-tidy drawback

- ideally, we want to maintain our custom check code within our code bases
A clang-tidy drawback

- ideally, we want to maintain our custom check code within our code bases
- clang-tidy is currently only **statically extensible**:
  - no real dynamic plugin mechanism foreseen
- new checks need (in principle) to be put **inside the clang-tidy source tree** … **tightly coupled** to the main llvm - clang git
- however … with a little work this can be circumvented …😊
“Hacked” plugin solution for clang-tidy

managed to decouple our check code from the main llvm/clang git

can compile this into a custom module which is picked up by clang-tidy (= real plugin mechanism)

llvm header + libs

clang header + libs

clang-tidy executable

can compile against

AliceO2 check module

our git
“Hacked” plugin solution for clang-tidy

managed to decouple our check code from the main llvm/clang git

can compile this into a custom module which is picked up by clang-tidy (= real plugin mechanism)

export LD_PRELOAD = libAliceO2Checks.so
clang-tidy --checks=-*,AliceO2* SourceFile.cxx
“Hacked” plugin solution for clang-tidy

managed to decouple our check code from the main llvm/clang git

can compile this into a custom module which is picked up by clang-tidy (= real plugin mechanism)

export LD_PRELOAD = libAliceO2Checks.so
clang-tidy --checks=-*,AliceO2* SourceFile.cxx

no free lunch: minimal code duplication from clang-tidy (“one header”) needs llvm shared libs installation
Development / usage status

- Implemented a few first custom checks (+fixes) for O2 to get us started
  - member/struct variable names, …  
    https://github.com/AliceO2Group/O2CodeChecker

- Use checks during pull request checking for O2 … based on alidist check recipe
  aliBuild build o2checkcode

- Made first use of automatic code fixing capabilities for C++11 guidelines
  (Giulio is having fun....)

- Gained experience integrating checks as unit-tests within CMake itself
Summary

❖ Very good first experience

❖ Can be easily exported / applied to other projects

❖ Need some manpower to implement more of our custom checks ....
Backup
Clang-tidy custom check example

```cpp
void MemberNamesCheck::registerMatchers(MatchFinder *Finder) {
    Finder->addMatcher(fieldDecl().bind("field_decl1"), this);
}

void MemberNamesCheck::check(const MatchFinder::MatchResult &Result) {
    const auto *MatchedDecl = Result.Nodes.getNodeAs<FieldDecl>("field_decl1");
    if (MatchedDecl) {
        // check that we are inside the Alice02 namespace to exclude system stuff
        // FIXME: needs to be configurable
        if (MatchedDecl->getQualifiedNameAsString().find("Alice02::") != 0)
            return;

        if (std::regex_match(MatchedDecl->getNameAsString(), Regex)) {
            return;
        }

diag(MatchedDecl->getLocation(),
        "field declaration %0 does not match naming rule",
        DiagnosticIDs::Error)
    } <<< MatchedDecl;
}
```

"register this check to act on C++ field declarations"

using the clang AST matcher concept

https://clang.llvm.org/docs/LibASTMatchers.html

check function is now invoked by framework for all field declarations

diagnostic output including source line etc.
Example for O2 checks

```cpp
namespace AliceO2 {
    class Foo {
        private:
            double mX;
            double fY;

            int mS = sizeof(int);
    }
};

./O2codecheck --checks=-*,*aliceO2* Foo.cxx -- -I ./ #compiler flags come here

/Users/swenzel/git/O2CodeChecker/build/tool/Foo.cxx:6:12: error: field declaration 'fY' does not match naming rule [aliceO2-member-name]
    double fY;
    ^
/Users/swenzel/git/O2CodeChecker/build/tool/Foo.cxx:8:14: warning: consider using sizeof() on instance instead on direct type [aliceO2-SizeOf]
    int mS = sizeof(int);
```
Overview of (demonstrator) tests implemented

❖ **Coding conventions**  [ see aliceO2 directory ]
  ❖ member name rule
  ❖ sizeof rule

❖ **Reporting tools**  [ see reporting directory ]
  ❖ scan code, collect and report a list of interfaces used for a particular class
  ❖ collect and report on virtual functions overloads