Sauberes C++ mit clang-tidy

Peter Hrenka

Linux Tag Tübingen 2017

24. Juni 2016

Über mich

- Linux Anwender seit 1995
- Studium Informatik und Mathematik in Tübingen
- Softwareentwickler `C++`, `python`, OpenGL
- regelmäßig auf OpenSource Konferenzen anzutreffen
- Programmiersprachenjunkie

Sauberer Code

```
#include <iostream>

int main(int argc, char* argv[])
{
  std::cout << "Clean Code" << std::endl;
  return 0;
}
```

Sauberer Code

```
#include <iostream>

int main(int argc, char* argv[])
{
  std::cout << "Clean␣Code" << std::endl;
  return 0;
}
```

```
"\n\n\n\n\n\n\n\n␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣\"\"#()*,0:::::;;<<<<<>
CC
[]
aaaaaaccccddddddeeeeeeegghiiiiilll
mmnnnnnnnooorrrrrrssstttttttuuuv
{}"
```

Sauberer Code

- Sauberkeit ist subjektiv

Sauberer Code

- Sauberkeit ist subjektiv
- Sauberkeit garantiert nicht
  - Fehlerfreiheit

Sauberer Code

- Sauberkeit ist subjektiv
- Sauberkeit garantiert nicht
  - Fehlerfreiheit
  - Sicherheit

Sauberer Code

- Sauberkeit ist subjektiv
- Sauberkeit garantiert nicht
  - Fehlerfreiheit
  - Sicherheit
  - Terminierung

Sauberer Code

- Sauberkeit ist subjektiv
- Sauberkeit garantiert nicht
  - Fehlerfreiheit
  - Sicherheit
  - Terminierung
  - Einhalten der Benutzererwartung

Sauberer Code

- Sauberkeit ist subjektiv
- Sauberkeit garantiert nicht
  - Fehlerfreiheit
  - Sicherheit
  - Terminierung
  - Einhalten der Benutzererwartung

Sauberer Code

- Sauberkeit ist subjektiv
- Sauberkeit garantiert nicht
  - Fehlerfreiheit
  - Sicherheit
  - Terminierung
  - Einhalten der Benutzererwartung
- Sauberkeit ist aber hilfreich für

Sauberer Code

- Sauberkeit ist subjektiv
- Sauberkeit garantiert nicht
  - Fehlerfreiheit
  - Sicherheit
  - Terminierung
  - Einhalten der Benutzererwartung
- Sauberkeit ist aber hilfreich für
  - Fehlersuche

Sauberer Code

- Sauberkeit ist subjektiv
- Sauberkeit garantiert nicht
  - Fehlerfreiheit
  - Sicherheit
  - Terminierung
  - Einhalten der Benutzererwartung
- Sauberkeit ist aber hilfreich für
  - Fehlersuche
  - Weiterentwicklung

Sauberer Code

- Sauberkeit ist subjektiv
- Sauberkeit garantiert nicht
  - Fehlerfreiheit
  - Sicherheit
  - Terminierung
  - Einhalten der Benutzererwartung
- Sauberkeit ist aber hilfreich für
  - Fehlersuche
  - Weiterentwicklung
  - Nachvollziehbarkeit

Sauberer Code

- Sauberkeit ist subjektiv
- Sauberkeit garantiert nicht
  - Fehlerfreiheit
  - Sicherheit
  - Terminierung
  - Einhalten der Benutzererwartung
- Sauberkeit ist aber hilfreich für
  - Fehlersuche
  - Weiterentwicklung
  - Nachvollziehbarkeit
  - Kooperation

Statische Analyse

- Programme nur anhand des Quelltextes analysieren

Statische Analyse

- Programme nur anhand des Quelltextes analysieren
- Implizit vom Compiler

Statische Analyse

- Programme nur anhand des Quelltextes analysieren
- Implizit vom Compiler
  - Warnungen

Statische Analyse

- Programme nur anhand des Quelltextes analysieren
- Implizit vom Compiler
  - Warnungen
  - Fehlermeldungen

Statische Analyse

- Programme nur anhand des Quelltextes analysieren
- Implizit vom Compiler
  - Warnungen
  - Fehlermeldungen
- Früher `lint` als Add-on zum Compiler

  `https://de.wikipedia.org/wiki/Lint_(Programmierwerkzeug)`

Statische Analyse

- Programme nur anhand des Quelltextes analysieren
- Implizit vom Compiler
  - Warnungen
  - Fehlermeldungen
- Früher `lint` als Add-on zum Compiler

  `https://de.wikipedia.org/wiki/Lint_(Programmierwerkzeug)`
- viele kommerzielle Anbieter von “Test-Tools”
  - Bauhaus

Statische Analyse

- Programme nur anhand des Quelltextes analysieren
- Implizit vom Compiler
  - Warnungen
  - Fehlermeldungen
- Früher `lint` als Add-on zum Compiler

  `https://de.wikipedia.org/wiki/Lint_(Programmierwerkzeug)`
- viele kommerzielle Anbieter von “Test-Tools”
  - Bauhaus
  - Covertiy

Statische Analyse

- Programme nur anhand des Quelltextes analysieren
- Implizit vom Compiler
  - Warnungen
  - Fehlermeldungen
- Früher `lint` als Add-on zum Compiler

  `https://de.wikipedia.org/wiki/Lint_(Programmierwerkzeug)`
- viele kommerzielle Anbieter von “Test-Tools”
  - Bauhaus
  - Covertiy
  - QA-C
- aber auch OpenSource Lösungen mit C++ Unterstützung

Statische Analyse

- Programme nur anhand des Quelltextes analysieren
- Implizit vom Compiler
  - Warnungen
  - Fehlermeldungen
- Früher `lint` als Add-on zum Compiler

  `https://de.wikipedia.org/wiki/Lint_(Programmierwerkzeug)`
- viele kommerzielle Anbieter von “Test-Tools”
  - Bauhaus
  - Covertiy
  - QA-C
- aber auch OpenSource Lösungen mit C++ Unterstützung
  - cpplint

Statische Analyse

- Programme nur anhand des Quelltextes analysieren
- Implizit vom Compiler
  - Warnungen
  - Fehlermeldungen
- Früher `lint` als Add-on zum Compiler

  `https://de.wikipedia.org/wiki/Lint_(Programmierwerkzeug)`
- viele kommerzielle Anbieter von “Test-Tools”
  - Bauhaus
  - Covertiy
  - QA-C
- aber auch OpenSource Lösungen mit C++ Unterstützung
  - cpplint
  - clang-tidy

Statische Analyse

- Programme nur anhand des Quelltextes analysieren
- Implizit vom Compiler
  - Warnungen
  - Fehlermeldungen
- Früher `lint` als Add-on zum Compiler

  `https://de.wikipedia.org/wiki/Lint_(Programmierwerkzeug)`
- viele kommerzielle Anbieter von “Test-Tools”
  - Bauhaus
  - Covertiy
  - QA-C
- aber auch OpenSource Lösungen mit C++ Unterstützung
  - cpplint
  - clang-tidy

clang-tidy

- Projekt innerhalb des llvm-Ökosystems `http://llvm.org`

clang-tidy

- Projekt innerhalb des llvm-Ökosystems `http://llvm.org`
- Unterproject von `clang`

clang-tidy

- Projekt innerhalb des llvm-Ökosystems `http://llvm.org`
- Unterproject von `clang`
- Eher unbekannt

clang-tidy

- Projekt innerhalb des llvm-Ökosystems `http://llvm.org`
- Unterproject von `clang`
- Eher unbekannt
- Dokumentation mit Verbesserungspotential

clang-tidy

- Projekt innerhalb des llvm-Ökosystems `http://llvm.org`
- Unterproject von `clang`
- Eher unbekannt
- Dokumentation mit Verbesserungspotential
- Callback Interface für Knoten des abstrakten Syntaxbaums (AST)

clang-tidy

- Projekt innerhalb des llvm-Ökosystems `http://llvm.org`
- Unterproject von `clang`
- Eher unbekannt
- Dokumentation mit Verbesserungspotential
- Callback Interface für Knoten des abstrakten Syntaxbaums (AST)
  → Mehrere Tests parallel durchführbar

clang-tidy

- Projekt innerhalb des llvm-Ökosystems `http://llvm.org`
- Unterproject von `clang`
- Eher unbekannt
- Dokumentation mit Verbesserungspotential
- Callback Interface für Knoten des abstrakten Syntaxbaums (AST)
  $\rightarrow$ Mehrere Tests parallel durchführbar
- Zugriff auf Preprozessor-Definitionen

clang-tidy

- Projekt innerhalb des llvm-Ökosystems `http://llvm.org`
- Unterproject von `clang`
- Eher unbekannt
- Dokumentation mit Verbesserungspotential
- Callback Interface für Knoten des abstrakten Syntaxbaums (AST)
  $\rightarrow$ Mehrere Tests parallel durchführbar
- Zugriff auf Preprozessor-Definitionen
- Auch Kommentare können ausgelesen werden

clang-tidy

- Projekt innerhalb des llvm-Ökosystems `http://llvm.org`
- Unterproject von `clang`
- Eher unbekannt
- Dokumentation mit Verbesserungspotential
- Callback Interface für Knoten des abstrakten Syntaxbaums (AST)
  $\rightarrow$ Mehrere Tests parallel durchführbar
- Zugriff auf Preprozessor-Definitionen
- Auch Kommentare können ausgelesen werden
- Kann Fix-Hints generieren

clang-tidy

- Projekt innerhalb des llvm-Ökosystems `http://llvm.org`
- Unterproject von `clang`
- Eher unbekannt
- Dokumentation mit Verbesserungspotential
- Callback Interface für Knoten des abstrakten Syntaxbaums (AST)
  → Mehrere Tests parallel durchführbar
- Zugriff auf Preprozessor-Definitionen
- Auch Kommentare können ausgelesen werden
- Kann Fix-Hints generieren
  → Automatische Korrekturen möglich

Installation

- Auf Debian, Ubuntu et al.

```
% sudo apt-get install clang-tidy-3.9
```

Vorbereitung

- `clang-tidy` benötigt Informationen über

Vorbereitung

- `clang-tidy` benötigt Informationen über
  - Includepfade (`-I/dir`)

Vorbereitung

- `clang-tidy` benötigt Informationen über
  - Includepfade (`-I/dir`)
  - Compileroptionen (z.B. `-std=c++11`)

Vorbereitung

- `clang-tidy` benötigt Informationen über
  - Includepfade (`-I/dir`)
  - Compileroptionen (z.B. `-std=c++11`)
  - Defines (`-DNDEBUG`)

Vorbereitung

- `clang-tidy` benötigt Informationen über
  - Includepfade (`-I/dir`)
  - Compileroptionen (z.B. `-std=c++11`)
  - Defines (`-DNDEBUG`)
- einfach auf Commandozeile nach `--` angeben

```
% clang-tidy-3.9 test.cpp -- I/dir -std=c++11 -DNDEBUG
```

- ... unpraktisch für größere Projekte

Vorbereitung CMake

- `CMake` kann notwendige Optionen für ein Projekt exportieren

Vorbereitung CMake

- CMake kann notwendige Optionen für ein Projekt exportieren
  `cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=ON /path/to/project`

Vorbereitung CMake

- CMake kann notwendige Optionen für ein Projekt exportieren
  `cmake -DCMAKE_EXPORT_COMPILE_COMMANDS=ON /path/to/project`
- erzeugt eine Compile-Database in einer Datei namens `compile_commands.json`
- Verwendbar mit allen Projekten, die CMake unterstützen
- Kein Build notwendig, `cmake`-Aufruf genügt

Aufrufoptionen für `clang-tidy`

- `-checks=` Liste von durchzuführenden Checks
  auch mit Wildcards, z.B. `llvm-*`
- `-fix` Eventuelle Fix-Vorschläge gleich auf Sourcen anwenden
- `-p` Pfad zur Compile-Database

Alle verfügbaren Checks anzeigen

```
% clang-tidy-3.9 -checks=* -list-checks
boost-use-to-string
cert-dcl03-c
cert-dcl50-cpp
cert-dcl54-cpp
cert-dcl59-cpp
cert-env33-c
cert-err34-c
cert-err52-cpp
cert-err58-cpp
cert-err60-cpp
cert-err61-cpp
cert-fio38-c
cert-flp30-c
cert-oop11-cpp
clang-analyzer-alpha.core.BoolAssignment
clang-analyzer-alpha.core.CallAndMessageUnInitRefArg
clang-analyzer-alpha.core.CastSize
clang-analyzer-alpha.core.CastToStruct
clang-analyzer-alpha.core.DynamicTypeChecker
clang-analyzer-alpha.core.FixedAddr
clang-analyzer-alpha.core.IdenticalExpr
clang-analyzer-alpha.core.PointerArithm
clang-analyzer-alpha.core.PointerSub
clang-analyzer-alpha.core.SizeofPtr
clang-analyzer-alpha.core.TestAfterDivZero
clang-analyzer-alpha.cplusplus.VirtualCall
```

```
clang-analyzer-alpha.deadcode.UnreachableCode
clang-analyzer-alpha.security.ArrayBound
clang-analyzer-alpha.security.ArrayBoundV2
clang-analyzer-alpha.security.MallocOverflow
clang-analyzer-alpha.security.ReturnPtrRange
clang-analyzer-alpha.security.taint.TaintPropagation
clang-analyzer-alpha.unix.Chroot
clang-analyzer-alpha.unix.PthreadLock
clang-analyzer-alpha.unix.SimpleStream
clang-analyzer-alpha.unix.Stream
clang-analyzer-alpha.unix.cstring.BufferOverlap
clang-analyzer-alpha.unix.cstring.NotNullTerminated
clang-analyzer-alpha.unix.cstring.OutOfBounds
clang-analyzer-core.CallAndMessage
clang-analyzer-core.DivideZero
clang-analyzer-core.DynamicTypePropagation
clang-analyzer-core.NonNullParamChecker
clang-analyzer-core.NullDereference
clang-analyzer-core.StackAddressEscape
clang-analyzer-core.UndefinedBinaryOperatorResult
clang-analyzer-core.VLASize
clang-analyzer-core.builtin.BuiltinFunctions
clang-analyzer-core.builtin.NoReturnFunctions
clang-analyzer-core.uninitialized.ArraySubscript
clang-analyzer-core.uninitialized.Assign
clang-analyzer-core.uninitialized.Branch
clang-analyzer-core.uninitialized.CapturedBlockVariable
clang-analyzer-core.uninitialized.UndefReturn
clang-analyzer-cplusplus.NewDelete
```

```
clang-analyzer-cplusplus.NewDeleteLeaks
clang-analyzer-deadcode.DeadStores
clang-analyzer-llvm.Conventions
clang-analyzer-nullability.NullPassedToNonnull
clang-analyzer-nullability.NullReturnedFromNonnull
clang-analyzer-nullability.NullableDereferenced
clang-analyzer-nullability.NullablePassedToNonnull
clang-analyzer-nullability.NullablePassedToNonnull
clang-analyzer-optin.mpi.MPI-Checker
clang-analyzer-optin.osx.cocoa.localizability.EmptyLocalizationContextChecker
clang-analyzer-optin.osx.cocoa.localizability.NonLocalizedStringChecker
clang-analyzer-optin.performance.Padding
clang-analyzer-security.FloatLoopCounter
clang-analyzer-security.insecureAPI.UncheckedReturn
clang-analyzer-security.insecureAPI.getpw
clang-analyzer-security.insecureAPI.gets
clang-analyzer-security.insecureAPI.mkstemp
clang-analyzer-security.insecureAPI.mktemp
clang-analyzer-security.insecureAPI.rand
clang-analyzer-security.insecureAPI.strcpy
clang-analyzer-security.insecureAPI.vfork
clang-analyzer-unix.API
clang-analyzer-unix.Malloc
clang-analyzer-unix.MallocSizeof
clang-analyzer-unix.MismatchedDeallocator
clang-analyzer-unix.Vfork
clang-analyzer-unix.cstring.BadSizeArg
clang-analyzer-unix.cstring.NullArg
cppcoreguidelines-c-copy-assignment-signature
```

```
cppcoreguidelines-interfaces-global-init
cppcoreguidelines-pro-bounds-array-to-pointer-decay
cppcoreguidelines-pro-bounds-constant-array-index
cppcoreguidelines-pro-bounds-pointer-arithmetic
cppcoreguidelines-pro-type-const-cast
cppcoreguidelines-pro-type-cstyle-cast
cppcoreguidelines-pro-type-member-init
cppcoreguidelines-pro-type-reinterpret-cast
cppcoreguidelines-pro-type-static-cast-downcast
cppcoreguidelines-pro-type-union-access
cppcoreguidelines-pro-type-vararg
google-build-explicit-make-pair
google-build-namespaces
google-build-using-namespace
google-default-arguments
google-explicit-constructor
google-global-names-in-headers
google-readability-braces-around-statements
google-readability-casting
google-readability-function-size
google-readability-namespace-comments
google-readability-redundant-smartptr-get
google-readability-todo
google-runtime-int
google-runtime-member-string-references
google-runtime-memset
google-runtime-operator
google-runtime-references
llvm-header-guard
```

```
llvm-include-order
llvm-namespace-comment
llvm-twine-local
misc-argument-comment
misc-assert-side-effect
misc-bool-pointer-implicit-conversion
misc-dangling-handle
misc-definitions-in-headers
misc-fold-init-type
misc-forward-declaration-namespace
misc-inaccurate-erase
misc-incorrect-roundings
misc-inefficient-algorithm
misc-macro-parentheses
misc-macro-repeated-side-effects
misc-misplaced-const
misc-misplaced-widening-cast
misc-move-const-arg
misc-move-constructor-init
misc-multiple-statement-macro
misc-new-delete-overloads
misc-noexcept-move-constructor
misc-non-copyable-objects
misc-pointer-and-integral-operation
misc-redundant-expression
misc-sizeof-container
misc-sizeof-expression
misc-static-assert
misc-string-constructor
```

```
misc-string-integer-assignment
misc-string-literal-with-embedded-nul
misc-suspicious-missing-comma
misc-suspicious-semicolon
misc-suspicious-string-compare
misc-swapped-arguments
misc-throw-by-value-catch-by-reference
misc-unconventional-assign-operator
misc-undelegated-constructor
misc-uniqueptr-reset-release
misc-unused-alias-decls
misc-unused-parameters
misc-unused-raii
misc-unused-using-decls
misc-virtual-near-miss
modernize-avoid-bind
modernize-deprecated-headers
modernize-loop-convert
modernize-make-shared
modernize-make-unique
modernize-pass-by-value
modernize-raw-string-literal
modernize-redundant-void-arg
modernize-replace-auto-ptr
modernize-shrink-to-fit
modernize-use-auto
modernize-use-bool-literals
modernize-use-default
modernize-use-emplace
```

```
modernize-use-nullptr
modernize-use-override
modernize-use-using
performance-faster-string-find
performance-for-range-copy
performance-implicit-cast-in-loop
performance-unnecessary-copy-initialization
performance-unnecessary-value-param
readability-avoid-const-params-in-decls
readability-braces-around-statements
readability-container-size-empty
readability-deleted-default
readability-else-after-return
readability-function-size
readability-identifier-naming
readability-implicit-bool-cast
readability-inconsistent-declaration-parameter-name
readability-named-parameter
readability-redundant-control-flow
readability-redundant-smartptr-get
readability-redundant-string-cstr
readability-redundant-string-init
readability-simplify-boolean-expr
readability-static-definition-in-anonymous-namespace
readability-uniqueptr-delete-release
```

Check-Gruppen 1 / 2

- C++ Core Guilelines
  `https://github.com/isocpp/CppCoreGuidelines/blob/master/CppCoreGuidelines.md`
- CERT Secure Coding Standards
  `https://www.securecoding.cert.org/confluence/display/seccode/SEI+CERT+Coding+Standards`
- Google C++ Style Guide
  `https://google.github.io/styleguide/cppguide.html`
- LLVM Style Guide
  `http://llvm.org/docs/CodingStandards.html`

Check-Gruppen 2 / 2

- `modernize-*` : Modernisierungen, ehemals `clang-modernize` Gelegenheiten Code durch modernere Sprachfeatures zu verbessern
- `performance-*` : Potentielle Performance-Verbesserungen
- `readablity-*` : Checks auf Lesbarkeit
- `misc-*` : Alles ohne bessere Kategorie

Beispiel

- Ein altes C++-Programm modernisieren

```
% clang-tidy -check=modernize-* -fix example.cpp -- -std=c++11
```

Beispiel, Originaldatei 1/2

```
#include <cstddef>  // for NULL
#include <memory>   // for auto_ptr

class Base {
public:
  Base() {}
  Base(int) {}
  virtual int method(int* val_ptr) const {
    return (val_ptr != NULL ? *val_ptr : 0);
  }
};
```

Beispiel, Originaldatei 2/2

```
class Derived : public Base {
public:
  int method(int* val_ptr) const {
    return (val_ptr != 0 ? *val_ptr : 1);
  }
};

int func()
{
  int iVal = 42;
  std::auto_ptr<Base>  base_hold(new Derived());

  return base_hold->method(&iVal);
}
```

Beispiel, Ausgabe 1/2

```
   119 warnings generated.
example.cpp:7:3: warning: use '= default' to define a trivial default constructor [modernize-use-default]
  Base() {}
         ^
         = default;
example.cpp:7:10: note: FIX-IT applied suggested code changes
  Base() {}
         ^
example.cpp:10:24: warning: use nullptr [modernize-use-nullptr]
    return (val_ptr != NULL ? *val_ptr : 0);
                       ^
                       nullptr
example.cpp:10:24: note: FIX-IT applied suggested code changes
    return (val_ptr != NULL ? *val_ptr : 0);
                       ^
example.cpp:16:7: warning: annotate this function with 'override' or (rarely) 'final' [modernize-use-override]
  int method(int* val_ptr) const {
                                 ^
                                 override
```

Beispiel, Ausgabe 2/2

```
example.cpp:16:33: note: FIX-IT applied suggested code changes
  int method(int* val_ptr) const {
                                ^
example.cpp:17:24: warning: use nullptr [modernize-use-nullptr]
    return (val_ptr != 0 ? *val_ptr : 1);
                       ^
                       nullptr
example.cpp:17:24: note: FIX-IT applied suggested code changes
    return (val_ptr != 0 ? *val_ptr : 1);
                       ^
example.cpp:24:8: warning: auto_ptr is deprecated, use unique_ptr instead [modernize-replace-auto-ptr]
  std::auto_ptr<Base>  base_hold(new Derived());
       ^
       unique_ptr
example.cpp:24:8: note: FIX-IT applied suggested code changes
  std::auto_ptr<Base>  base_hold(new Derived());
       ^
clang-tidy applied 5 of 5 suggested fixes.
Suppressed 114 warnings (114 in non-user code).
Use -header-filter=.* to display errors from all non-system headers.
```

Beispiel, modernisierte Datei 1/2

```
#include <cstddef>  // for NULL
#include <memory>   // for auto_ptr

class Base {
public:
  Base() = default;
  Base(int) {}
  virtual int method(int* val_ptr) const {
    return (val_ptr != nullptr ? *val_ptr : 0);
  }
};
```

Beispiel, modernisierte Datei 2/2

```
class Derived : public Base {
public:
  int method(int* val_ptr) const override {
    return (val_ptr != nullptr ? *val_ptr : 1);
  }
};

int func()
{
  int iVal = 42;
  std::unique_ptr<Base>  base_hold(new Derived());

  return base_hold->method(&iVal);
}
```

Modernisierungen

- `clang-tidy` versteht Vererbung
- Kann leere Implementierungen erkennen
- Kann erkennen, wann `0` als Integer oder Pointer gebraucht wird
- Das ist nur schwer mit `sed` oder `awk` zu emulieren

Installation

- Akutellen Quelltext aus git-Mirror herunterladen
- Brauche jeweils `llvm`, `clang` und `clang-tools-extra`

```
% git clone http://llvm.org/git/llvm.git
% cd llvm/tools
% git clone http://llvm.org/git/clang.git
% cd clang/tools
% git clone http://llvm.org/git/clang-tools-extra.git extra
```

- Sourcen ca. 1,1 GB Plattenplatz
- Build ca. 3 GB Plattenplatz

Bauen

- Eingebettet in `llvm`-build
  - Etwas Schade, weil alles mitgebaut werden muss
- Aufruf etwa

```
% mkdir build && cd build && cmake --build ../llvm
```

- Dauer:

Bauen

- Eingebettet in `llvm`-build
  - Etwas Schade, weil alles mitgebaut werden muss
- Aufruf etwa

```
% mkdir build && cd build && cmake --build ../llvm
```

- Dauer: viel zu lang

Eigenen Check ausdenken

- Irgendwas Nützliches

Eigenen Check ausdenken

- Irgendwas Nützliches
- Nicht zu kompliziert

Eigenen Check ausdenken

- Irgendwas Nützliches
- Nicht zu kompliziert
- Also:

Eigenen Check ausdenken

- Irgendwas Nützliches
- Nicht zu kompliziert
- Also:
  - Floating-Pointe Literale (z.B. `0.0`) ohne Suffix sind immer `double`
  - Manche (wenige) Entwickler wissen das nicht
  - Potentiell Probleme wegen implizieter Typ-Konvertierung
  - Wir wollen uns nicht darauf verlassen, das der Compiler alles richtig macht
  - Alle Float-Literale sollen einen expliziten cast erhalten:

    `0.0` → `static_cast<double>(0.0)`
  - Semantisch keine Änderung

Eigenen Check implementieren

- Früher: Anpassen vieler Dateien

Eigenen Check implementieren

- Früher: Anpassen vieler Dateien
- Heute: Python-Tool zum Anlegen eines neuen Tests

```
% ./clang-tidy/add_new_check.py misc explicit-double
Updating ./clang-tidy/misc/CMakeLists.txt...
Creating ./clang-tidy/misc/ExplicitDoubleCheck.h...
Creating ./clang-tidy/misc/ExplicitDoubleCheck.cpp...
Updating ./clang-tidy/misc/MiscTidyModule.cpp...
Creating test/clang-tidy/misc-explicit-double.cpp...
Creating docs/clang-tidy/checks/misc-explicit-double.rst...
Updating docs/clang-tidy/checks/list.rst...
Done. Now it's your turn!
```

Generierter Header

```
#define LLVM_CLANG_TOOLS_EXTRA_CLANG_TIDY_MISC_EXPLICIT_DOUBLE_H

#include "../ClangTidy.h"

namespace clang {
namespace tidy {
namespace misc {

/// FIXME: Write a short description.
///
/// For the user-facing documentation see:
/// http://clang.llvm.org/extra/clang-tidy/checks/misc-explicit-double.html
class ExplicitDoubleCheck : public ClangTidyCheck {
public:
  ExplicitDoubleCheck(StringRef Name, ClangTidyContext *Context)
      : ClangTidyCheck(Name, Context) {}
  void registerMatchers(ast_matchers::MatchFinder *Finder) override;
  void check(const ast_matchers::MatchFinder::MatchResult &Result) override;
};

} // namespace misc
} // namespace tidy
} // namespace clang

#endif // LLVM_CLANG_TOOLS_EXTRA_CLANG_TIDY_MISC_EXPLICIT_DOUBLE_H
```

Generierter Source

```
#include "ExplicitDoubleCheck.h"
#include "clang/AST/ASTContext.h"
#include "clang/ASTMatchers/ASTMatchFinder.h"

using namespace clang::ast_matchers;

namespace clang {
namespace tidy {
namespace misc {

void ExplicitDoubleCheck::registerMatchers(MatchFinder *Finder) {
  // FIXME: Add matchers.
  Finder->addMatcher(functionDecl().bind("x"), this);
}

void ExplicitDoubleCheck::check(const MatchFinder::MatchResult &Result) {
  // FIXME: Add callback implementation.
  const auto *MatchedDecl = Result.Nodes.getNodeAs<FunctionDecl>("x");
  if (MatchedDecl->getName().startswith("awesome_"))
    return;
  diag(MatchedDecl->getLocation(), "function␣%0␣is␣insufficiently␣awesome")
      << MatchedDecl
      << FixItHint::CreateInsertion(MatchedDecl->getLocation(), "awesome_");
}

} // namespace misc
```

API verstehen

- Konsultiere Online-Dokumentation
- `http://clang.llvm.org/docs/LibASTMatchers.html`
- `http://clang.llvm.org/docs/LibASTMatchersReference.html#decl-matchers`
- `http://clang.llvm.org/doxygen/classclang_1_1FloatingLiteral.html`

Unsere Implementierung

```
void ExplicitDoubleCheck::check(const MatchFinder::MatchResult &Result) {
  const auto *MatchedFloat = Result.Nodes.getNodeAs<FloatingLiteral>("floatLit");

  // ignore non double literals
  if (MatchedFloat->getType()->isSpecificBuiltinType(clang::BuiltinType::Double) == false) {
    diag(MatchedFloat->getLocation(), "␣found␣non-double␣float␣literal");
    return;
  }

  SourceLocation EndLoc = Lexer::getLocForEndOfToken( MatchedFloat->getLocEnd(),
                                                      0, *Result.SourceManager,
                                                      getLangOpts());

  diag(EndLoc, "␣found␣double␣literal")
    << FixItHint::CreateInsertion(MatchedFloat->getLocation(),"static_cast<double>(")
    << FixItHint::CreateInsertion(EndLoc, ")");
}
```

Beispiel

```
float f_offset = 1.0f;
double offset = 42.0;
long double l_offset = 100.0l;

double add_abs(double inputVal=1.0)
{
  if (inputVal >= 0.0) {
    return (inputVal + offset);
  } else {
    return (inputVal - offset);
  }
}
```

Ausgabe 1 / 2

```
./bin/clang-tidy -checks=misc-explicit-double -fix ../doubles.cpp --
5 warnings generated.
build/../doubles.cpp:2:18: warning:  found non-double float literal [misc-explicit-double]
float f_offset = 1.0f;
                 ^
build/../doubles.cpp:3:21: warning:  found double literal [misc-explicit-double]
double offset = 42.0;
                ^
                static_cast<double>( )
build/../doubles.cpp:3:17: note: FIX-IT applied suggested code changes
double offset = 42.0;
                ^
build/../doubles.cpp:3:21: note: FIX-IT applied suggested code changes
double offset = 42.0;
                    ^
build/../doubles.cpp:4:24: warning:  found non-double float literal [misc-explicit-double]
long double l_offset = 100.01;
                       ^
build/../doubles.cpp:6:35: warning:  found double literal [misc-explicit-double]
double add_abs(double inputVal=1.0)
                                  ^
                                  static_cast<double>( )
```

Ausgabe 2 / 2

```
build/../doubles.cpp:6:32: note: FIX-IT applied suggested code changes
double add_abs(double inputVal=1.0)
                               ^
build/../doubles.cpp:6:35: note: FIX-IT applied suggested code changes
double add_abs(double inputVal=1.0)
                                  ^
build/../doubles.cpp:8:22: warning:  found double literal [misc-explicit-double]
  if (inputVal >= 0.0) {
                   ^
                    static_cast<double>( )
build/../doubles.cpp:8:19: note: FIX-IT applied suggested code changes
  if (inputVal >= 0.0) {
                  ^
build/../doubles.cpp:8:22: note: FIX-IT applied suggested code changes
  if (inputVal >= 0.0) {
                     ^
clang-tidy applied 6 of 6 suggested fixes.
```

Gefixted Beispiel

```
float f_offset = 1.0f;
double offset = static_cast<double>(42.0);
long double l_offset = 100.0l;

double add_abs(double inputVal=static_cast<double>(1.0))
{
  if (inputVal >= static_cast<double>(0.0)) {
    return (inputVal + offset);
  } else {
    return (inputVal - offset);
  }
}
```

Fazit

+ Mächtiges Tool zur Überprüfung von C++-Code
+ Modernisierungen interessant für Updates auf C++11
  - Leider kaum Features für C++14, C++17 enthalten
+ Vernünftige C++11 API von `llvm` und `clang`
- Build-Prozess sehr monolitisch, nur zusammen mit `llvm`
- Checks relativ langsam

Weitere interessante Projekte

- `scan-build` : Tool von `clang` um Teile der `clang-tidy` Checks auszuführen
- `valgrind` : Bitgenaue Emulation mit Definiertheitstracking, Speicherchecker
- Sanitize-Optionen mit Laufzeitchecks in `GCC` und `clang`
  - `-fsanitize=undefined` : Checks auf undefiniertes Verhalten
  - `-fsanitize=signed-integer-overflow` : Prüfe auf Überlauf von Integern
- `klee` : Symbolische Ausführung von LLVM Code
- `American Fuzz Lop` : Instrumentalisierender Fuzzer

Vielen Dank!

Fragen?