# Vector klase

Generated by Doxygen 1.13.2

# Chapter 1

# 3.0 nuosavos Vector klasės testavimas

## 1.1 Aprašymas

Ši Vector<T> klasė yra sukurta siekiant atkartoti `std::vector` elgseną. Ji palaiko didžiąją dalį funkcionalumo, įskaitant dinaminius atminties pokyčius, operatorius, iteratorius ir kt. Testuota naudojant `doctest` ir lyginta su `std::vector`.

## 1.2 5 funkcijų pavyzdžiai

### 1.2.1 1. operator[]

```
reference operator[](size_type pos) {
    return vec_[pos];
}
```

```
Vector<int> v = {1, 2, 3};
std::cout « v[1]; // Output: 2
```

### 1.2.2 2. push_back

```
void push_back(const T& value) {
    if (size_ >= capacity_) {
        reserve(capacity_ == 0 ? 1 : capacity_ * 2);
    }
    vec_[size_++] = value;
}
```

```
Vector<std::string> v;
v.push_back("labas");
v.push_back("pasauli");
```

### 1.2.3 3. at()

```
reference at(size_type pos) {
    if (pos >= size_) {
        throw std::out_of_range("out of range");
}
 return vec_[pos];
}
```

```
Vector<int> v = {10, 20};
try {
    v.at(5);
} catch (const std::out_of_range& e) {
    std::cout « "Klaida: " « e.what();
}
```

### 1.2.4 4. `operator==`

```
bool operator==(const Vector& other) const {
    if (size_ != other.size_) return false;
    return std::equal(begin(), end(), other.begin());
}
Vector<int> a = {1, 2, 3};
Vector<int> b = {1, 2, 3};
std::cout « (a == b); // Output: 1
```

### 1.2.5 5. `empty()`

```
bool empty() const noexcept {
    return size_ == 0;
}
if (numbers.empty()) {
    std::cout « "Vector yra tuščias!\n";
}

numbers.push_back(10);

if (!numbers.empty()) {
    std::cout « "Vector jau nėra tuščias!\n";
}
```

## 1.3 Testavimas

Klasė testuota su `doctest`, testuojant:

- Konstruktorius ir priskyrimus
- Elementų prieigą
- Iteracijas ir atminties valdymą
- Modifikatorius (`push_back`, `erase`, `insert`, `resize` ir kt.)
- Operatorius (==, !=, <, > ir kt.)

## 1.4 Efektyvumo analizė

Toliau pateikiama palyginamoji `push_back()` operacijos trukmė skirtingiems elementų kiekiams, matuojant vidutinį laiką milisekundėmis:

| **Elementų kiekis** | **`std::vector` (ms)** | **`Vector` (ms)** |
|---|---:|---:|
| 10000 | 0.217 | 0.069 |
| 100000 | 1.85 | 0.555 |
| 1000000 | 18.597 | 4.959 |
| 10000000 | 189.844 | 54.176 |
| 100000000 | 1745.370 | 462.541 |

**Komentaras:**

- Mūsų `Vector` klasė ženkliai spartesnė nei `std::vector` pagal šiuos bandymus.

## 1.5 Perskirstymų skaičius

# Chapter 2

# Hierarchical Index

## 2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 3

# Class Index

## 3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 4

# File Index

## 4.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 5

# Class Documentation

## 5.1 doctest::Approx Struct Reference

### Public Member Functions

- **Approx** (double value)
- Approx **operator()** (double value) const
- Approx & **epsilon** (double newEpsilon)
- Approx & **scale** (double newScale)

### Public Attributes

- double **m_epsilon**
- double **m_scale**
- double **m_value**

### Friends

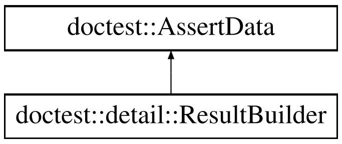
- DOCTEST_INTERFACE friend bool **operator==** (double lhs, const Approx &rhs)
- DOCTEST_INTERFACE friend bool **operator==** (const Approx &lhs, double rhs)
- DOCTEST_INTERFACE friend bool **operator!=** (double lhs, const Approx &rhs)
- DOCTEST_INTERFACE friend bool **operator!=** (const Approx &lhs, double rhs)
- DOCTEST_INTERFACE friend bool **operator<=** (double lhs, const Approx &rhs)
- DOCTEST_INTERFACE friend bool **operator<=** (const Approx &lhs, double rhs)
- DOCTEST_INTERFACE friend bool **operator>=** (double lhs, const Approx &rhs)
- DOCTEST_INTERFACE friend bool **operator>=** (const Approx &lhs, double rhs)
- DOCTEST_INTERFACE friend bool **operator<** (double lhs, const Approx &rhs)
- DOCTEST_INTERFACE friend bool **operator<** (const Approx &lhs, double rhs)
- DOCTEST_INTERFACE friend bool **operator>** (double lhs, const Approx &rhs)
- DOCTEST_INTERFACE friend bool **operator>** (const Approx &lhs, double rhs)

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.2 doctest::AssertData Struct Reference

Inheritance diagram for doctest::AssertData:



### Classes

- class StringContains

### Public Member Functions

- **AssertData** (assertType::Enum at, const char ∗file, int line, const char ∗expr, const char ∗exception_type, const StringContains &exception_string)

### Public Attributes

- const TestCaseData ∗ **m_test_case**
- assertType::Enum **m_at**
- const char ∗ **m_file**
- int **m_line**
- const char ∗ **m_expr**
- bool **m_failed**
- bool **m_threw**
- String **m_exception**
- String **m_decomp**
- bool **m_threw_as**
- const char ∗ **m_exception_type**
- class DOCTEST_INTERFACE doctest::AssertData::StringContains **m_exception_string**

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.3 std::basic_istream< charT, traits > Class Template Reference

The documentation for this class was generated from the following file:

- Testavimas/doctest.h

## 5.4 std::basic_ostream< charT, traits > Class Template Reference

The documentation for this class was generated from the following file:

- Testavimas/doctest.h

## 5.5 std::char_traits< charT > Struct Template Reference

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.6 doctest::Contains Class Reference

### Public Member Functions

- **Contains** (const String &string)
- bool **checkWith** (const String &other) const

### Public Attributes

- String **string**

The documentation for this class was generated from the following file:

- Testavimas/doctest.h

## 5.7 doctest::Context Class Reference

### Public Member Functions

- **Context** (int argc=0, const char ∗const ∗argv=nullptr)
- **Context** (const Context &)=delete
- **Context** (Context &&)=delete
- Context & **operator=** (const Context &)=delete
- Context & **operator=** (Context &&)=delete
- void **applyCommandLine** (int argc, const char ∗const ∗argv)
- void **addFilter** (const char ∗filter, const char ∗value)
- void **clearFilters** ()
- void **setOption** (const char ∗option, bool value)
- void **setOption** (const char ∗option, int value)
- void **setOption** (const char ∗option, const char ∗value)
- bool **shouldExit** ()
- void **setAsDefaultForAssertsOutOfTestCases** ()
- void **setAssertHandler** (detail::assert_handler ah)
- void **setCout** (std::ostream ∗out)
- int **run** ()

The documentation for this class was generated from the following file:

- Testavimas/doctest.h

## 5.8 doctest::ContextOptions Struct Reference

OCLINT too many fields.

```
#include <doctest.h>
```

### Public Attributes

- std::ostream ∗ **cout** = nullptr
- String **binary_name**
- const detail::TestCase ∗ **currentTest** = nullptr
- String **out**
- String **order_by**
- unsigned **rand_seed**
- unsigned **first**
- unsigned **last**
- int **abort_after**
- int **subcase_filter_levels**
- bool **success**
- bool **case_sensitive**
- bool **exit**
- bool **duration**
- bool **minimal**
- bool **quiet**
- bool **no_throw**
- bool **no_exitcode**
- bool **no_run**
- bool **no_intro**
- bool **no_version**
- bool **no_colors**
- bool **force_colors**
- bool **no_breaks**
- bool **no_skip**
- bool **gnu_file_line**
- bool **no_path_in_filenames**
- String **strip_file_prefixes**
- bool **no_line_numbers**
- bool **no_debug_output**
- bool **no_skipped_summary**
- bool **no_time_in_output**
- bool **help**
- bool **version**
- bool **count**
- bool **list_test_cases**
- bool **list_test_suites**
- bool **list_reporters**

### 5.8.1 Detailed Description

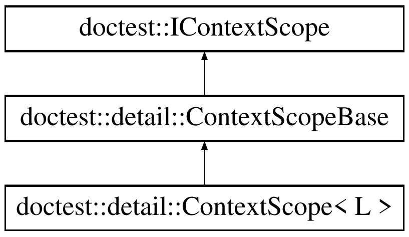
OCLINT too many fields.

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.9 doctest::detail::ContextScope< L > Class Template Reference

Inheritance diagram for doctest::detail::ContextScope< L >:


doctest::IContextScope
doctest::detail::ContextScopeBase
doctest::detail::ContextScope< L >

#### Public Member Functions

- **ContextScope** (const L &lambda)
- **ContextScope** (L &&lambda)
- **ContextScope** (const ContextScope &)=delete
- **ContextScope** (ContextScope &&) noexcept=default
- ContextScope & **operator=** (const ContextScope &)=delete
- ContextScope & **operator=** (ContextScope &&)=delete
- void stringify (std::ostream *s) const override

### Public Member Functions inherited from doctest::detail::ContextScopeBase

- **ContextScopeBase** (const ContextScopeBase &)=delete
- ContextScopeBase & **operator=** (const ContextScopeBase &)=delete
- ContextScopeBase & **operator=** (ContextScopeBase &&)=delete

#### Additional Inherited Members

### Protected Member Functions inherited from doctest::detail::ContextScopeBase

- **ContextScopeBase** (ContextScopeBase &&other) noexcept
- void **destroy** ()

### Protected Attributes inherited from doctest::detail::ContextScopeBase

- bool **need_to_destroy** {true}

### 5.9.1 Member Function Documentation

#### 5.9.1.1 stringify()

```
template<typename L>
void doctest::detail::ContextScope< L >::stringify (
            std::ostream * s) const  [inline], [override], [virtual]
```

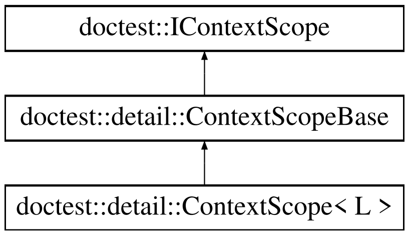
Implements doctest::IContextScope.

The documentation for this class was generated from the following file:

- Testavimas/doctest.h

## 5.10 doctest::detail::ContextScopeBase Struct Reference

Inheritance diagram for doctest::detail::ContextScopeBase:


doctest::IContextScope
doctest::detail::ContextScopeBase
doctest::detail::ContextScope< L >

#### Public Member Functions

- **ContextScopeBase** (const ContextScopeBase &)=delete
- ContextScopeBase & **operator=** (const ContextScopeBase &)=delete
- ContextScopeBase & **operator=** (ContextScopeBase &&)=delete

### Public Member Functions inherited from doctest::IContextScope

- virtual void **stringify** (std::ostream ∗) const =0

#### Protected Member Functions

- **ContextScopeBase** (ContextScopeBase &&other) noexcept
- void **destroy** ()

#### Protected Attributes

- bool **need_to_destroy** {true}

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.11 doctest::CurrentTestCaseStats Struct Reference

#### Public Attributes

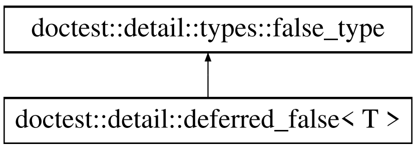
- int **numAssertsCurrentTest**
- int **numAssertsFailedCurrentTest**
- double **seconds**
- int **failure_flags**
- bool **testCaseSuccess**

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.12 doctest::detail::deferred_false< T > Struct Template Reference

Inheritance diagram for doctest::detail::deferred_false< T >:


doctest::detail::types::false_type
doctest::detail::deferred_false< T >

#### Additional Inherited Members

### Static Public Attributes inherited from doctest::detail::types::false_type

- static DOCTEST_CONSTEXPR bool **value** = false

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.13 doctest::detail::types::enable_if< COND, T > Struct Template Reference

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.14 doctest::detail::types::enable_if< true, T > Struct Template Reference

#### Public Types

- using **type** = T

The documentation for this struct was generated from the following file:

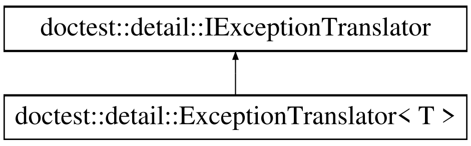
- Testavimas/doctest.h

## 5.15 doctest::detail::ExceptionTranslator< T > Class Template Reference

OCLINT destructor of virtual class.

```
#include <doctest.h>
```

Inheritance diagram for doctest::detail::ExceptionTranslator< T >:


doctest::detail::IExceptionTranslator
doctest::detail::ExceptionTranslator< T >

### Public Member Functions

- **ExceptionTranslator** (String(∗translateFunction)(T))
- bool translate (String &res) const override

### 5.15.1 Detailed Description

**template<typename T>**
**class doctest::detail::ExceptionTranslator< T >**

OCLINT destructor of virtual class.

### 5.15.2 Member Function Documentation

#### 5.15.2.1 translate()

```
template<typename T>
bool doctest::detail::ExceptionTranslator< T >::translate (
            String & res) const  [inline], [override], [virtual]
```

Implements doctest::detail::IExceptionTranslator.

The documentation for this class was generated from the following file:

- Testavimas/doctest.h

## 5.16 doctest::detail::Expression_lhs< L > Struct Template Reference

### Public Member Functions

- **Expression_lhs** (L &&in, assertType::Enum at)
- DOCTEST_NOINLINE operator Result ()
- **operator L** () const

### Public Attributes

- L **lhs**
- assertType::Enum **m_at**

## 5.16.1 Member Function Documentation

### 5.16.1.1 operator Result()

```
template<typename L>
DOCTEST_NOINLINE doctest::detail::Expression_lhs< L >::operator Result ()  [inline]
```

OCLINT bitwise operator in conditional

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.17 doctest::detail::ExpressionDecomposer Struct Reference

### Public Member Functions

- **ExpressionDecomposer** (assertType::Enum at)
- template< typename L >
  Expression_lhs< const L && > **operator**<< (const L &&operand)
- template< typename L, typename types::enable_if< !doctest::detail::types::is_rvalue_reference< L >::value, void >::type ∗ = nullptr>
  Expression_lhs< const L & > **operator**<< (const L &operand)

### Public Attributes

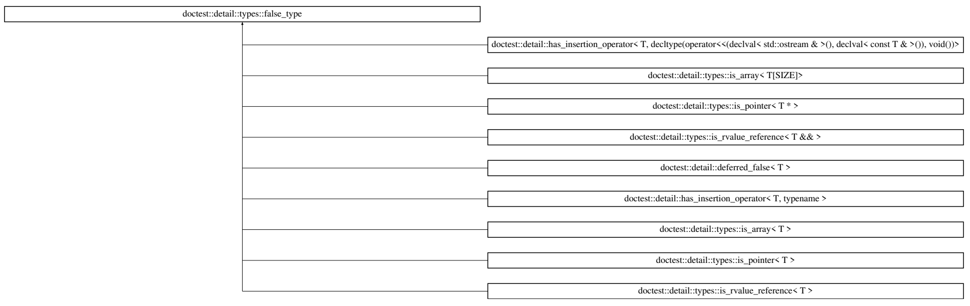
- assertType::Enum **m_at**

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.18 doctest::detail::types::false_type Struct Reference

Inheritance diagram for doctest::detail::types::false_type:


doctest::detail::types::false_type
doctest::detail::has_insertion_operator< T, decltype(operator<<(declval< std::ostream & >(), declval< const T & >()), void())>
doctest::detail::types::is_array< T[SIZE]>
doctest::detail::types::is_pointer< T * >
doctest::detail::types::is_rvalue_reference< T && >
doctest::detail::deferred_false< T >
doctest::detail::has_insertion_operator< T, typename >
doctest::detail::types::is_array< T >
doctest::detail::types::is_pointer< T >
doctest::detail::types::is_rvalue_reference< T >

### Static Public Attributes

- static DOCTEST_CONSTEXPR bool **value** = false

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.19 doctest::detail::filldata< T > Struct Template Reference

### Static Public Member Functions

- static void **fill** (std::ostream ∗stream, const T &in)

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.20 doctest::detail::filldata< const char[N]> Struct Template Reference

### Static Public Member Functions

- static void **fill** (std::ostream ∗stream, const char(&in)[N])
- static void **fill** (std::ostream ∗stream, const const char &in)

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.21 doctest::detail::filldata< const void ∗ > Struct Reference

### Static Public Member Functions

- static void **fill** (std::ostream ∗stream, const void ∗in)
- static void **fill** (std::ostream ∗stream, const const void ∗&in)

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.22 doctest::detail::filldata< T * > Struct Template Reference

### Static Public Member Functions

- static void **fill** (std::ostream *stream, const T *in)
- static void **fill** (std::ostream *stream, const T &in)

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.23 doctest::detail::filldata< T[N]> Struct Template Reference

### Static Public Member Functions

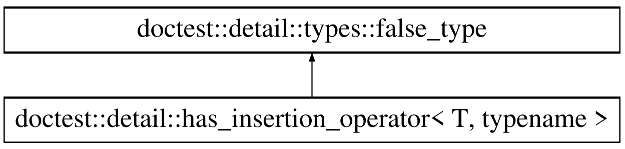
- static void **fill** (std::ostream *stream, const T(&in)[N])
- static void **fill** (std::ostream *stream, const T &in)

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.24 doctest::detail::has_insertion_operator< T, typename > Struct Template Reference

Inheritance diagram for doctest::detail::has_insertion_operator< T, typename >:


doctest::detail::types::false_type
doctest::detail::has_insertion_operator< T, typename >

### Additional Inherited Members

### Static Public Attributes inherited from doctest::detail::types::false_type

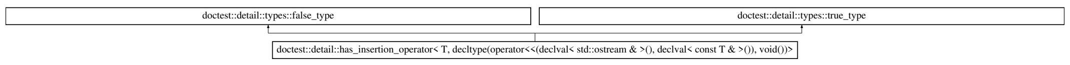
- static DOCTEST_CONSTEXPR bool **value** = false

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.25 doctest::detail::has_insertion_operator< T, decltype(operator<<(declval< std::ostream & >(), declval< const T & >()), void())> Struct Template Reference

Inheritance diagram for doctest::detail::has_insertion_operator< T, decltype(operator<<(declval< std::ostream & >(), declval< const T & >()), void())>:


doctest::detail::types::false_type
doctest::detail::types::true_type
doctest::detail::has_insertion_operator< T, decltype(operator<<(declval< std::ostream & >(), declval< const T & >()), void())>

**Additional Inherited Members**

### Static Public Attributes inherited from doctest::detail::types::false_type

- static DOCTEST_CONSTEXPR bool **value** = false

### Static Public Attributes inherited from doctest::detail::types::true_type

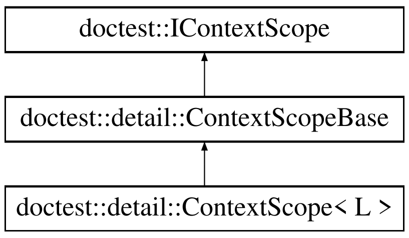
- static DOCTEST_CONSTEXPR bool **value** = true

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.26 doctest::IContextScope Struct Reference

Inheritance diagram for doctest::IContextScope:


doctest::IContextScope
doctest::detail::ContextScopeBase
doctest::detail::ContextScope< L >

**Public Member Functions**

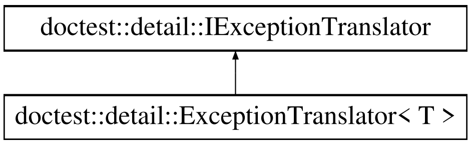
- virtual void **stringify** (std::ostream ∗) const =0

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.27 doctest::detail::IExceptionTranslator Struct Reference

Inheritance diagram for doctest::detail::IExceptionTranslator:


doctest::detail::IExceptionTranslator
doctest::detail::ExceptionTranslator< T >

### Public Member Functions

- virtual bool **translate** (String &) const =0

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.28 doctest::IReporter Struct Reference

### Public Member Functions

- virtual void **report_query** (const QueryData &)=0
- virtual void **test_run_start** ()=0
- virtual void **test_run_end** (const TestRunStats &)=0
- virtual void **test_case_start** (const TestCaseData &)=0
- virtual void **test_case_reenter** (const TestCaseData &)=0
- virtual void **test_case_end** (const CurrentTestCaseStats &)=0
- virtual void **test_case_exception** (const TestCaseException &)=0
- virtual void **subcase_start** (const SubcaseSignature &)=0
- virtual void **subcase_end** ()=0
- virtual void **log_assert** (const AssertData &)=0
- virtual void **log_message** (const MessageData &)=0
- virtual void **test_case_skipped** (const TestCaseData &)=0

### Static Public Member Functions

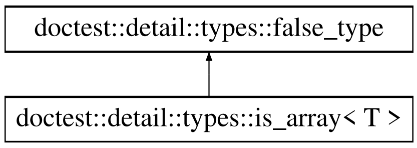
- static int **get_num_active_contexts** ()
- static const IContextScope $*$const $*$ **get_active_contexts** ()
- static int **get_num_stringified_contexts** ()
- static const String $*$ **get_stringified_contexts** ()

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.29 doctest::detail::types::is_array< T > Struct Template Reference

Inheritance diagram for doctest::detail::types::is_array< T >:


doctest::detail::types::false_type
doctest::detail::types::is_array< T >

#### Additional Inherited Members

### Static Public Attributes inherited from doctest::detail::types::false_type

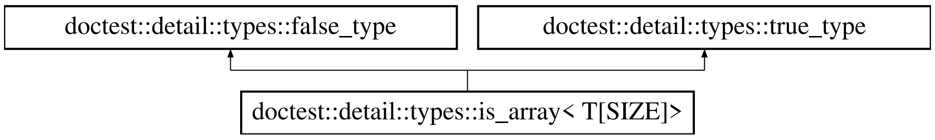
- static DOCTEST_CONSTEXPR bool **value** = false

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.30 doctest::detail::types::is_array< T[SIZE]> Struct Template Reference

Inheritance diagram for doctest::detail::types::is_array< T[SIZE]>:


doctest::detail::types::false_type
doctest::detail::types::true_type
doctest::detail::types::is_array< T[SIZE]>

#### Additional Inherited Members

### Static Public Attributes inherited from doctest::detail::types::false_type

- static DOCTEST_CONSTEXPR bool **value** = false

### Static Public Attributes inherited from doctest::detail::types::true_type

- static DOCTEST_CONSTEXPR bool **value** = true

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.31 doctest::detail::types::is_enum< T > Struct Template Reference

### Static Public Attributes

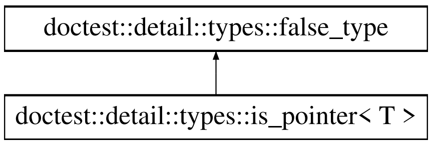
- static DOCTEST_CONSTEXPR bool **value** = __is_enum(T)

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.32 doctest::detail::types::is_pointer< T > Struct Template Reference

Inheritance diagram for doctest::detail::types::is_pointer< T >:


doctest::detail::types::false_type
doctest::detail::types::is_pointer< T >

### Additional Inherited Members

### Static Public Attributes inherited from doctest::detail::types::false_type

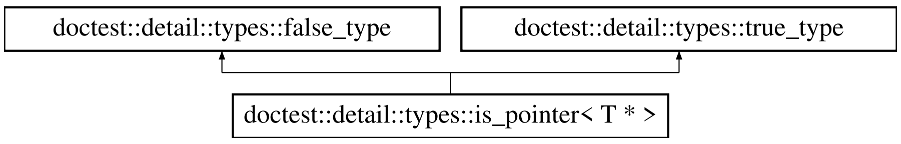
- static DOCTEST_CONSTEXPR bool **value** = false

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.33 doctest::detail::types::is_pointer< T * > Struct Template Reference

Inheritance diagram for doctest::detail::types::is_pointer< T * >:


doctest::detail::types::false_type
doctest::detail::types::true_type
doctest::detail::types::is_pointer< T * >

### Additional Inherited Members

### Static Public Attributes inherited from doctest::detail::types::false_type

- static DOCTEST_CONSTEXPR bool **value** = false

### Static Public Attributes inherited from doctest::detail::types::true_type

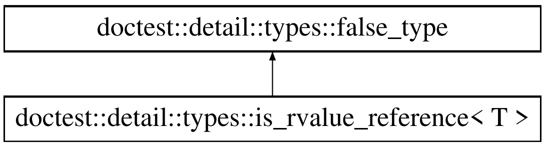
- static DOCTEST_CONSTEXPR bool **value** = true

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.34 doctest::detail::types::is_rvalue_reference< T > Struct Template Reference

Inheritance diagram for doctest::detail::types::is_rvalue_reference< T >:


doctest::detail::types::false_type
doctest::detail::types::is_rvalue_reference< T >

#### Additional Inherited Members

### Static Public Attributes inherited from doctest::detail::types::false_type

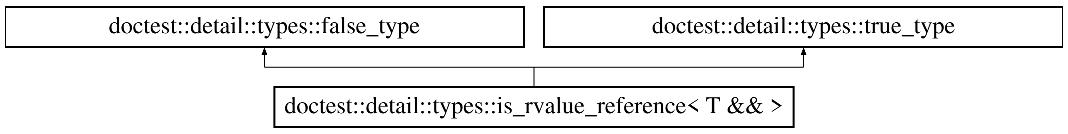
- static DOCTEST_CONSTEXPR bool **value** = false

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.35 doctest::detail::types::is_rvalue_reference< T && > Struct Template Reference

Inheritance diagram for doctest::detail::types::is_rvalue_reference< T && >:


doctest::detail::types::false_type
doctest::detail::types::true_type
doctest::detail::types::is_rvalue_reference< T && >

#### Additional Inherited Members

### Static Public Attributes inherited from doctest::detail::types::false_type

- static DOCTEST_CONSTEXPR bool **value** = false

### Static Public Attributes inherited from doctest::detail::types::true_type

- static DOCTEST_CONSTEXPR bool **value** = true

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.36 doctest::IsNaN< F > Struct Template Reference

### Public Member Functions

- **IsNaN** (F f, bool flip=false)
- IsNaN< F > **operator!** () const
- **operator bool** () const

### Public Attributes

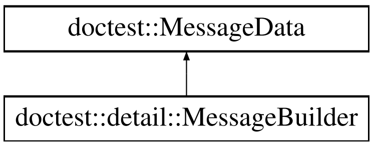
- F **value**
- bool **flipped**

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.37 doctest::detail::MessageBuilder Struct Reference

Inheritance diagram for doctest::detail::MessageBuilder:



### Public Member Functions

- **MessageBuilder** (const char ∗file, int line, assertType::Enum severity)
- **MessageBuilder** (const MessageBuilder &)=delete
- **MessageBuilder** (MessageBuilder &&)=delete
- MessageBuilder & **operator=** (const MessageBuilder &)=delete
- MessageBuilder & **operator=** (MessageBuilder &&)=delete
- template<typename T>
  MessageBuilder & **operator,** (const T &in)
- template<typename T>
  DOCTEST_MSVC_SUPPRESS_WARNING_POP MessageBuilder & **operator**<< (const T &in)
- template<typename T>
  MessageBuilder & **operator**∗ (const T &in)
- bool **log** ()
- void **react** ()

**Public Attributes**

- std::ostream ∗ **m_stream**
- bool **logged** = false

### Public Attributes inherited from doctest::MessageData

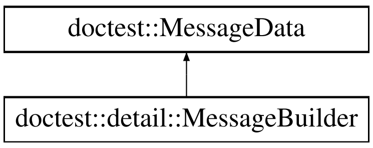
- String **m_string**
- const char ∗ **m_file**
- int **m_line**
- assertType::Enum **m_severity**

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.38 doctest::MessageData Struct Reference

Inheritance diagram for doctest::MessageData:



**Public Attributes**

- String **m_string**
- const char ∗ **m_file**
- int **m_line**
- assertType::Enum **m_severity**

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.39 doctest::QueryData Struct Reference

**Public Attributes**

- const TestRunStats ∗ **run_stats** = nullptr
- const TestCaseData ∗∗ **data** = nullptr
- unsigned **num_data** = 0

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.40 doctest::detail::RelationalComparator< int, L, R > Struct Template Reference

**Public Member Functions**

- bool **operator()** (const DOCTEST_REF_WRAP(L), const DOCTEST_REF_WRAP(R)) const

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.41 doctest::detail::types::remove_const< T > Struct Template Reference

**Public Types**

- using **type** = T

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.42 doctest::detail::types::remove_const< const T > Struct Template Reference

**Public Types**

- using **type** = T
- using **type**

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.43 doctest::detail::types::remove_reference< T > Struct Template Reference

**Public Types**

- using **type** = T

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.44 doctest::detail::types::remove_reference< T & > Struct Template Reference

#### Public Types

- using **type** = T
- using **type**

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.45 doctest::detail::types::remove_reference< T && > Struct Template Reference

#### Public Types

- using **type** = T
- using **type**

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.46 doctest::detail::Result Struct Reference

#### Public Member Functions

- **Result** (bool passed, const String &decomposition=String())

#### Public Attributes

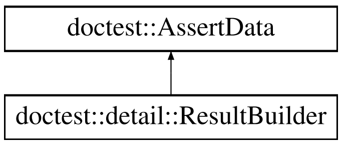
- bool **m_passed**
- String **m_decomp**

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.47 doctest::detail::ResultBuilder Struct Reference

Inheritance diagram for doctest::detail::ResultBuilder:



### Public Member Functions

- **ResultBuilder** (assertType::Enum at, const char ∗file, int line, const char ∗expr, const char ∗exception_↩ type="", const String &exception_string="")
- **ResultBuilder** (assertType::Enum at, const char ∗file, int line, const char ∗expr, const char ∗exception_type, const Contains &exception_string)
- void **setResult** (const Result &res)
- template<int comparison, typename L, typename R>
  DOCTEST_NOINLINE bool **binary_assert** (const DOCTEST_REF_WRAP(L) lhs, const DOCTEST_REF↩ _WRAP(R) rhs)
- template<typename L>
  DOCTEST_NOINLINE bool unary_assert (const DOCTEST_REF_WRAP(L) val)
- void **translateException** ()
- bool **log** ()
- void **react** () const

### Public Member Functions inherited from doctest::AssertData

- **AssertData** (assertType::Enum at, const char ∗file, int line, const char ∗expr, const char ∗exception_type, const StringContains &exception_string)

### Additional Inherited Members

### Public Attributes inherited from doctest::AssertData

- const TestCaseData ∗ **m_test_case**
- assertType::Enum **m_at**
- const char ∗ **m_file**
- int **m_line**
- const char ∗ **m_expr**
- bool **m_failed**
- bool **m_threw**
- String **m_exception**
- String **m_decomp**
- bool **m_threw_as**
- const char ∗ **m_exception_type**
- class DOCTEST_INTERFACE doctest::AssertData::StringContains **m_exception_string**

### 5.47.1 Member Function Documentation

#### 5.47.1.1 unary_assert()

```
template<typename L>
DOCTEST_NOINLINE bool doctest::detail::ResultBuilder::unary_assert (
            const DOCTEST_REF_WRAP(L) val)  [inline]
```

OCLINT bitwise operator in conditional

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.48 doctest::detail::should_stringify_as_underlying_type< T > Struct Template Reference

### Static Public Attributes

- static DOCTEST_CONSTEXPR bool **value** = detail::types::is_enum<T>::value && !doctest::detail::has_insertion_operator<T
  ::value

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.49 doctest::String Class Reference

### Public Types

- using **size_type** = DOCTEST_CONFIG_STRING_SIZE_TYPE

### Public Member Functions

- **String** (const char *in)
- **String** (const char *in, size_type in_size)
- **String** (std::istream &in, size_type in_size)
- **String** (const String &other)
- String & **operator=** (const String &other)
- String & **operator+=** (const String &other)
- **String** (String &&other) noexcept
- String & **operator=** (String &&other) noexcept
- char **operator[ ]** (size_type i) const
- char & **operator[ ]** (size_type i)
- const char * **c_str** () const
- char * **c_str** ()
- size_type **size** () const
- size_type **capacity** () const
- String **substr** (size_type pos, size_type cnt=npos) &&
- String **substr** (size_type pos, size_type cnt=npos) const &
- size_type **find** (char ch, size_type pos=0) const
- size_type **rfind** (char ch, size_type pos=npos) const
- int **compare** (const char *other, bool no_case=false) const
- int **compare** (const String &other, bool no_case=false) const

### Static Public Attributes

- static DOCTEST_CONSTEXPR size_type **npos** = static_cast<size_type>(-1)

### Friends

- DOCTEST_INTERFACE std::ostream & **operator**<< (std::ostream &s, const String &in)

The documentation for this class was generated from the following file:

- Testavimas/doctest.h

## 5.50 doctest::AssertData::StringContains Class Reference

### Public Member Functions

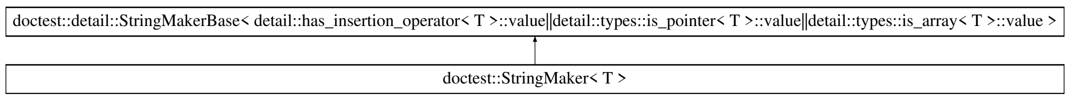
- **StringContains** (const String &str)
- **StringContains** (Contains cntn)
- bool **check** (const String &str)
- **operator const String &** () const
- const char ∗ **c_str** () const

The documentation for this class was generated from the following file:

- Testavimas/doctest.h

## 5.51 doctest::StringMaker< T > Struct Template Reference

Inheritance diagram for doctest::StringMaker< T >:


doctest::detail::StringMakerBase< detail::has_insertion_operator< T >::value||detail::types::is_pointer< T >::value||detail::types::is_array< T >::value >
doctest::StringMaker< T >

### Additional Inherited Members

### Static Public Member Functions inherited from doctest::detail::StringMakerBase< detail::has_insertion_operator< T >::value||detail::types::is_pointe

- static String **convert** (const DOCTEST_REF_WRAP(T))

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.52 doctest::detail::StringMakerBase< C > Struct Template Reference

### Static Public Member Functions

- template<typename T>
  static String **convert** (const DOCTEST_REF_WRAP(T))

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.53 doctest::detail::StringMakerBase< true > Struct Reference

### Static Public Member Functions

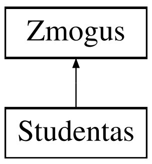
- template<typename T>
  static String **convert** (const DOCTEST_REF_WRAP(T) in)
- static String **convert** (const DOCTEST_REF_WRAP(T))

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.54 Studentas Class Reference

Inheritance diagram for Studentas:



### Public Member Functions

- **Studentas** (const std::string &vardas, const std::string &pavarde, const Vector< int > &nd, int egzaminas)
- **Studentas** (const Studentas &other)
- **Studentas** (Studentas &&other) noexcept
- Studentas & **operator=** (const Studentas &other)
- Studentas & **operator=** (Studentas &&other) noexcept
- int **egzaminas** () const
- double **galutinis** () const
- const Vector< int > & **nd** () const
- void **setEgzaminas** (int egzaminas)
- void **pridetiND** (int pazymys)
- void **skaiciuotiGalutini** (char metodas)
- void **generuotiPazymius** (int kiek)
- std::istream & **read** (std::istream &is)
- std::ostream & spausdinti (std::ostream &os) const override

### Public Member Functions inherited from Zmogus

- **Zmogus** (const std::string &vardas, const std::string &pavarde)
- std::string **vardas** () const
- std::string **pavarde** () const
- void **setVardas** (const std::string &vardas)
- void **setPavarde** (const std::string &pavarde)

#### Friends

- std::ostream & **operator**<< (std::ostream &os, const Studentas &s)
- std::istream & **operator**>> (std::istream &is, Studentas &s)
- bool **compareVardas** (const Studentas &a, const Studentas &b)
- bool **comparePavarde** (const Studentas &a, const Studentas &b)
- bool **compareGalutinis** (const Studentas &a, const Studentas &b)

#### Additional Inherited Members

### Protected Attributes inherited from Zmogus

- std::string **vardas_**
- std::string **pavarde_**

### 5.54.1 Member Function Documentation

#### 5.54.1.1 spausdinti()

```
std::ostream & Studentas::spausdinti (
            std::ostream & os) const  [override], [virtual]
```

Implements Zmogus.

The documentation for this class was generated from the following files:

- studentas.h
- studentas.cpp

## 5.55 doctest::detail::Subcase Struct Reference

#### Public Member Functions

- **Subcase** (const String &name, const char *file, int line)
- **Subcase** (const Subcase &)=delete
- **Subcase** (Subcase &&)=delete
- Subcase & **operator=** (const Subcase &)=delete
- Subcase & **operator=** (Subcase &&)=delete
- **operator bool** () const

#### Public Attributes

- SubcaseSignature **m_signature**
- bool **m_entered** = false

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.56 doctest::SubcaseSignature Struct Reference

#### Public Member Functions

- bool **operator==** (const SubcaseSignature &other) const
- bool **operator<** (const SubcaseSignature &other) const

#### Public Attributes

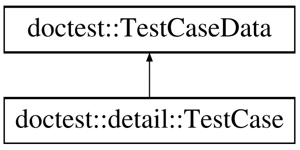
- String **m_name**
- const char ∗ **m_file**
- int **m_line**

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.57 doctest::detail::TestCase Struct Reference

Inheritance diagram for doctest::detail::TestCase:



#### Public Member Functions

- **TestCase** (funcType test, const char ∗file, unsigned line, const TestSuite &test_suite, const String &type=String(), int template_id=-1)
- **TestCase** (const TestCase &other)
- **TestCase** (TestCase &&)=delete
- TestCase & **operator=** (const TestCase &other)
- DOCTEST_MSVC_SUPPRESS_WARNING_POP TestCase & **operator=** (TestCase &&)=delete
- TestCase & **operator∗** (const char ∗in)
- template<typename T>
  TestCase & **operator∗** (const T &in)
- bool **operator<** (const TestCase &other) const

**Public Attributes**

- funcType **m_test**
- String **m_type**
- int **m_template_id**
- String **m_full_name**

### Public Attributes inherited from doctest::TestCaseData

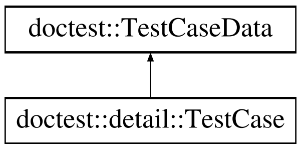
- String **m_file**
- unsigned **m_line**
- const char ∗ **m_name**
- const char ∗ **m_test_suite**
- const char ∗ **m_description**
- bool **m_skip**
- bool **m_no_breaks**
- bool **m_no_output**
- bool **m_may_fail**
- bool **m_should_fail**
- int **m_expected_failures**
- double **m_timeout**

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.58 doctest::TestCaseData Struct Reference

Inheritance diagram for doctest::TestCaseData:



**Public Attributes**

- String **m_file**
- unsigned **m_line**
- const char ∗ **m_name**
- const char ∗ **m_test_suite**
- const char ∗ **m_description**
- bool **m_skip**
- bool **m_no_breaks**
- bool **m_no_output**
- bool **m_may_fail**
- bool **m_should_fail**
- int **m_expected_failures**
- double **m_timeout**

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.59 doctest::TestCaseException Struct Reference

### Public Attributes

- String **error_string**
- bool **is_crash**

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.60 doctest::detail::TestFailureException Struct Reference

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.61 doctest::TestRunStats Struct Reference

### Public Attributes

- unsigned **numTestCases**
- unsigned **numTestCasesPassingFilters**
- unsigned **numTestSuitesPassingFilters**
- unsigned **numTestCasesFailed**
- int **numAsserts**
- int **numAssertsFailed**

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.62 doctest::detail::TestSuite Struct Reference

### Public Member Functions

- TestSuite & **operator*** (const char *in)
- template<typename T>
  TestSuite & **operator*** (const T &in)

#### Public Attributes

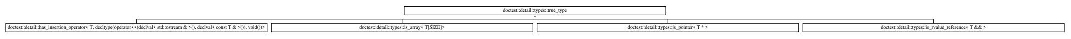
- const char ∗ **m_test_suite** = nullptr
- const char ∗ **m_description** = nullptr
- bool **m_skip** = false
- bool **m_no_breaks** = false
- bool **m_no_output** = false
- bool **m_may_fail** = false
- bool **m_should_fail** = false
- int **m_expected_failures** = 0
- double **m_timeout** = 0

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.63 doctest::detail::types::true_type Struct Reference

Inheritance diagram for doctest::detail::types::true_type:


doctest::detail::types::true_type
doctest::detail::has_insertion_operator< T, decltype(operator<<(declval< std::ostream & >(), declval< const T & >()), void())>
doctest::detail::types::is_array< T[SIZE]>
doctest::detail::types::is_pointer< T * >
doctest::detail::types::is_rvalue_reference< T && >

#### Static Public Attributes

- static DOCTEST_CONSTEXPR bool **value** = true

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.64 std::tuple< Types > Class Template Reference

The documentation for this class was generated from the following file:

- Testavimas/doctest.h

## 5.65 doctest::detail::types::underlying_type< T > Struct Template Reference

#### Public Types

- using **type** = __underlying_type(T)

The documentation for this struct was generated from the following file:

- Testavimas/doctest.h

## 5.66 Vector< T > Class Template Reference

### Public Types

- using **value_type** = T
- using **size_type** = size_t
- using **reference** = T&
- using **const_reference** = const T&
- using **iterator** = T∗
- using **const_iterator** = const T∗
- using **reverse_iterator** = std::reverse_iterator<iterator>
- using **const_reverse_iterator** = std::reverse_iterator<const_iterator>
- using **value_type** = T
- using **size_type** = size_t
- using **reference** = T&
- using **const_reference** = const T&
- using **iterator** = T∗
- using **const_iterator** = const T∗
- using **reverse_iterator** = std::reverse_iterator<iterator>
- using **const_reverse_iterator** = std::reverse_iterator<const_iterator>

### Public Member Functions

- **Vector** (size_type count)
- **Vector** (size_type count, const T &value)
- **Vector** (std::initializer_list< T > list)
- **Vector** (const Vector &other)
- **Vector** (Vector &&other) noexcept
- Vector & **operator=** (const Vector &other)
- Vector & **operator=** (Vector &&other) noexcept
- reference **operator[ ]** (size_type pos)
- const_reference **operator[ ]** (size_type pos) const
- reference **at** (size_type pos)
- const_reference **at** (size_type pos) const
- reference **front** ()
- const_reference **front** () const
- reference **back** ()
- const_reference **back** () const
- T ∗ **data** () noexcept
- const T ∗ **data** () const noexcept
- iterator **begin** () noexcept
- const_iterator **begin** () const noexcept
- const_iterator **cbegin** () const noexcept
- iterator **end** () noexcept
- const_iterator **end** () const noexcept
- const_iterator **cend** () const noexcept
- reverse_iterator **rbegin** () noexcept
- const_reverse_iterator **rbegin** () const noexcept
- const_reverse_iterator **crbegin** () const noexcept
- reverse_iterator **rend** () noexcept
- const_reverse_iterator **rend** () const noexcept
- const_reverse_iterator **crend** () const noexcept
- bool **empty** () const noexcept

- size_type **size** () const noexcept
- size_type **capacity** () const noexcept
- size_type **getReallocations** () const
- size_type **max_size** () const noexcept
- void **reserve** (size_type new_cap)
- void **shrink_to_fit** ()
- void **clear** () noexcept
- iterator **insert** (const_iterator pos, const T &value)
- iterator **insert** (const_iterator pos, T &&value)
- iterator **erase** (const_iterator pos)
- iterator **erase** (const_iterator first, const_iterator last)
- void **push_back** (const T &value)
- void **push_back** (T &&value)
- template<typename... Args>
  reference **emplace_back** (Args &&... args)
- template<typename... Args>
  iterator **emplace** (const_iterator pos, Args &&... args)
- void **pop_back** ()
- void **resize** (size_type count)
- void **resize** (size_type count, const value_type &value)
- void **swap** (Vector &other) noexcept
- bool **operator==** (const Vector &other) const
- bool **operator!=** (const Vector &other) const
- bool **operator<** (const Vector &other) const
- bool **operator<=** (const Vector &other) const
- bool **operator>** (const Vector &other) const
- bool **operator>=** (const Vector &other) const
- **Vector** (size_type count)
- **Vector** (size_type count, const T &value)
- **Vector** (std::initializer_list< T > list)
- **Vector** (const Vector &other)
- **Vector** (Vector &&other) noexcept
- Vector & **operator=** (const Vector &other)
- Vector & **operator=** (Vector &&other) noexcept
- reference **operator[ ]** (size_type pos)
- const_reference **operator[ ]** (size_type pos) const
- reference **at** (size_type pos)
- const_reference **at** (size_type pos) const
- reference **front** ()
- const_reference **front** () const
- reference **back** ()
- const_reference **back** () const
- T ∗ **data** () noexcept
- const T ∗ **data** () const noexcept
- iterator **begin** () noexcept
- const_iterator **begin** () const noexcept
- const_iterator **cbegin** () const noexcept
- iterator **end** () noexcept
- const_iterator **end** () const noexcept
- const_iterator **cend** () const noexcept
- reverse_iterator **rbegin** () noexcept
- const_reverse_iterator **rbegin** () const noexcept
- const_reverse_iterator **crbegin** () const noexcept
- reverse_iterator **rend** () noexcept
- const_reverse_iterator **rend** () const noexcept

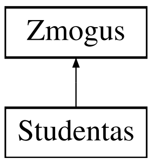
- const_reverse_iterator **crend** () const noexcept
- bool **empty** () const noexcept
- size_type **size** () const noexcept
- size_type **capacity** () const noexcept
- size_type **getReallocations** () const
- size_type **max_size** () const noexcept
- void **reserve** (size_type new_cap)
- void **shrink_to_fit** ()
- void **clear** () noexcept
- iterator **insert** (const_iterator pos, const T &value)
- iterator **insert** (const_iterator pos, T &&value)
- iterator **erase** (const_iterator pos)
- iterator **erase** (const_iterator first, const_iterator last)
- void **push_back** (const T &value)
- void **push_back** (T &&value)
- template<typename... Args>
  reference **emplace_back** (Args &&... args)
- template<typename... Args>
  iterator **emplace** (const_iterator pos, Args &&... args)
- void **pop_back** ()
- void **resize** (size_type count)
- void **resize** (size_type count, const value_type &value)
- void **swap** (Vector &other) noexcept
- bool **operator==** (const Vector &other) const
- bool **operator!=** (const Vector &other) const
- bool **operator<** (const Vector &other) const
- bool **operator<=** (const Vector &other) const
- bool **operator>** (const Vector &other) const
- bool **operator>=** (const Vector &other) const

The documentation for this class was generated from the following files:

- Testavimas/vector.h
- vector.h

## 5.67 Zmogus Class Reference

Inheritance diagram for Zmogus:



### Public Member Functions

- **Zmogus** (const std::string &vardas, const std::string &pavarde)
- std::string **vardas** () const
- std::string **pavarde** () const
- void **setVardas** (const std::string &vardas)
- void **setPavarde** (const std::string &pavarde)
- virtual std::ostream & **spausdinti** (std::ostream &os) const =0

**Protected Attributes**

- std::string **vardas_**
- std::string **pavarde_**

The documentation for this class was generated from the following file:

- zmogus.h

# Chapter 6

# File Documentation

## 6.1 funkcijos.h

```
#ifndef FUNKCIJOS_H
#define FUNKCIJOS_H

#include "Studentas.h"
#include <string>
#include "vector.h"
#include <iostream>
#include <fstream>
#include <algorithm>
#include <iomanip>
#include <cstdlib>
#include <ctime>
#include <cctype>
#include <sstream>
#include <limits>
#include <stdexcept>
#include <chrono>
#include <vector>

// Funkcijos, kurios dirba su studentų sąrašu

bool arTinkamasVardas(const std::string& tekstas);
bool arTinkamasPazymys(int& pazymys);

double skaiciuotiVidurki(const Vector<int>& pazymiai);
double skaiciuotiMediana(Vector<int> pazymiai);

// Dirba su vienu Studentas objektu
void skaiciuotiGalutiniBala(Studentas& studentas, char metodas);
void generuotiPazymius(Studentas& studentas, int kiek);

// Dirba su studentų vektoriumi
void generuotiStudentus(Vector<Studentas>& studentai, int kiek, int ndSk);
void spausdintiRezultatus(const Vector<Studentas>& studentai, std::ostream& out);
void nuskaitytiIsFailo(Vector<Studentas>& studentai, const std::string& failoVardas);
void rikiuotiStudentus(Vector<Studentas>& studentai, char kriterijus);
void generuotiFaila(const std::string& failoPavadinimas, int studentuKiekis, int ndSk, char metodas);
void padalintiStudentus(const Vector<Studentas>& studentai, Vector<Studentas>& vargsiai,
      Vector<Studentas>& kietiakai);
void spausdintiStudentusIFaila(const Vector<Studentas>& studentai, const std::string& failoVardas);
void apdorotiFaila(const std::string& failoVardas, char metodas);

void vykdytiPrograma(); // Pagrindinė funkcija programos valdymui

#endif
```

## 6.2 studentas.h

```
#ifndef STUDENTAS_H
#define STUDENTAS_H

#include <ostream>
#include <iostream>
```

```cpp
#include <string>
#include "vector.h"
#include <algorithm>
#include <numeric>
#include "zmogus.h"

class Studentas : public Zmogus {
private:
    Vector<int> nd_;
    int egzaminas_;
    double galutinis_;

public:
    Studentas(); // tuščias konstruktorius
    Studentas(const std::string& vardas, const std::string& pavarde, const Vector<int>& nd, int
egzaminas);
    Studentas(const Studentas& other);            // Copy constructor
    Studentas(Studentas&& other) noexcept;        // Move constructor
    ~Studentas();                                 // Destruktorius
    Studentas& operator=(const Studentas& other); // Copy assignment
    Studentas& operator=(Studentas&& other) noexcept; // Move assignment

    // Getteriai
    int egzaminas() const;
    double galutinis() const;
    const Vector<int>& nd() const;

    // Setteriai
    void setEgzaminas(int egzaminas);
    void pridetiND(int pazymys);

    // Metodai
    void skaiciuotiGalutini(char metodas);
    void generuotiPazymius(int kiek);
    std::istream& read(std::istream& is);
    std::ostream& spausdinti(std::ostream& os) const override;

    // I/O operatoriai
    friend std::ostream& operator«(std::ostream& os, const Studentas& s);
    friend std::istream& operator»(std::istream& is, Studentas& s);

    // Friend funkcijos palyginimui
    friend bool compareVardas(const Studentas& a, const Studentas& b);
    friend bool comparePavarde(const Studentas& a, const Studentas& b);
    friend bool compareGalutinis(const Studentas& a, const Studentas& b);
};

// Deklaracijos palyginimui
bool compareVardas(const Studentas& a, const Studentas& b);
bool comparePavarde(const Studentas& a, const Studentas& b);
bool compareGalutinis(const Studentas& a, const Studentas& b);
#endif
```

## 6.3 doctest.h

```cpp
// ====================================================================== lgtm
[cpp/missing-header-guard]
// == DO NOT MODIFY THIS FILE BY HAND - IT IS AUTO GENERATED BY CMAKE! ==
// ======================================================================
//
// doctest.h - the lightest feature-rich C++ single-header testing framework for unit tests and TDD
//
// Copyright (c) 2016-2023 Viktor Kirilov
//
// Distributed under the MIT Software License
// See accompanying file LICENSE.txt or copy at
// https://opensource.org/licenses/MIT
//
// The documentation can be found at the library's page:
// https://github.com/doctest/doctest/blob/master/doc/markdown/readme.md
//
// =================================================================================================
// =================================================================================================
// =================================================================================================
//
// The library is heavily influenced by Catch - https://github.com/catchorg/Catch2
// which uses the Boost Software License - Version 1.0
// see here - https://github.com/catchorg/Catch2/blob/master/LICENSE.txt
//
// The concept of subcases (sections in Catch) and expression decomposition are from there.
// Some parts of the code are taken directly:
// - stringification - the detection of "ostream& operator«(ostream&, const T&)" and StringMaker<>
// - the Approx() helper class for floating point comparison
// - colors in the console
// - breaking into a debugger
// - signal / SEH handling
// - timer
// - XmlWriter class - thanks to Phil Nash for allowing the direct reuse (AKA copy/paste)
//
// The expression decomposing templates are taken from lest - https://github.com/martinmoene/lest
// which uses the Boost Software License - Version 1.0
// see here - https://github.com/martinmoene/lest/blob/master/LICENSE.txt
//
// =================================================================================================
// =================================================================================================
// =================================================================================================

#ifndef DOCTEST_LIBRARY_INCLUDED
#define DOCTEST_LIBRARY_INCLUDED

// =================================================================================================
// == VERSION ======================================================================================
// =================================================================================================

#define DOCTEST_VERSION_MAJOR 2
#define DOCTEST_VERSION_MINOR 4
#define DOCTEST_VERSION_PATCH 12

// util we need here
#define DOCTEST_TOSTR_IMPL(x) #x
#define DOCTEST_TOSTR(x) DOCTEST_TOSTR_IMPL(x)

#define DOCTEST_VERSION_STR                                                                        \
    DOCTEST_TOSTR(DOCTEST_VERSION_MAJOR) "."                                                       \
    DOCTEST_TOSTR(DOCTEST_VERSION_MINOR) "."                                                       \
    DOCTEST_TOSTR(DOCTEST_VERSION_PATCH)

#define DOCTEST_VERSION                                                                            \
    (DOCTEST_VERSION_MAJOR * 10000 + DOCTEST_VERSION_MINOR * 100 + DOCTEST_VERSION_PATCH)

// =================================================================================================
// == COMPILER VERSION =============================================================================
// =================================================================================================

// ideas for the version stuff are taken from here: https://github.com/cxxstuff/cxx_detect

#ifdef _MSC_VER
#define DOCTEST_CPLUSPLUS _MSVC_LANG
#else
#define DOCTEST_CPLUSPLUS __cplusplus
#endif

#define DOCTEST_COMPILER(MAJOR, MINOR, PATCH) ((MAJOR)*10000000 + (MINOR)*100000 + (PATCH))

// GCC/Clang and GCC/MSVC are mutually exclusive, but Clang/MSVC are not because of clang-cl...
#if defined(_MSC_VER) && defined(_MSC_FULL_VER)
#if _MSC_VER == _MSC_FULL_VER / 10000
#define DOCTEST_MSVC DOCTEST_COMPILER(_MSC_VER / 100, _MSC_VER % 100, _MSC_FULL_VER % 10000)
#else // MSVC
#define DOCTEST_MSVC                                                                               \
    DOCTEST_COMPILER(_MSC_VER / 100, (_MSC_FULL_VER / 100000) % 100, _MSC_FULL_VER % 100000)
#endif // MSVC
#endif // MSVC
#if defined(__clang__) && defined(__clang_minor__) && defined(__clang_patchlevel__)
#define DOCTEST_CLANG DOCTEST_COMPILER(__clang_major__, __clang_minor__, __clang_patchlevel__)
#elif defined(__GNUC__) && defined(__GNUC_MINOR__) && defined(__GNUC_PATCHLEVEL__) &&              \
        !defined(__INTEL_COMPILER)
#define DOCTEST_GCC DOCTEST_COMPILER(__GNUC__, __GNUC_MINOR__, __GNUC_PATCHLEVEL__)
#endif // GCC
#if defined(__INTEL_COMPILER)
#define DOCTEST_ICC DOCTEST_COMPILER(__INTEL_COMPILER / 100, __INTEL_COMPILER % 100, 0)
#endif // ICC

#ifndef DOCTEST_MSVC
#define DOCTEST_MSVC 0
#endif // DOCTEST_MSVC
#ifndef DOCTEST_CLANG
#define DOCTEST_CLANG 0
#endif // DOCTEST_CLANG
#ifndef DOCTEST_GCC
#define DOCTEST_GCC 0
#endif // DOCTEST_GCC
#ifndef DOCTEST_ICC
#define DOCTEST_ICC 0
#endif // DOCTEST_ICC

// =================================================================================================
// == COMPILER WARNINGS HELPERS ====================================================================
// =================================================================================================

#if DOCTEST_CLANG && !DOCTEST_ICC
#define DOCTEST_PRAGMA_TO_STR(x) _Pragma(#x)
#define DOCTEST_CLANG_SUPPRESS_WARNING_PUSH _Pragma("clang diagnostic push")
#define DOCTEST_CLANG_SUPPRESS_WARNING(w) DOCTEST_PRAGMA_TO_STR(clang diagnostic ignored w)
#define DOCTEST_CLANG_SUPPRESS_WARNING_POP _Pragma("clang diagnostic pop")
#define DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH(w)                                                \
    DOCTEST_CLANG_SUPPRESS_WARNING_PUSH DOCTEST_CLANG_SUPPRESS_WARNING(w)
#else // DOCTEST_CLANG
#define DOCTEST_CLANG_SUPPRESS_WARNING_PUSH
#define DOCTEST_CLANG_SUPPRESS_WARNING(w)
#define DOCTEST_CLANG_SUPPRESS_WARNING_POP
#define DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH(w)
#endif // DOCTEST_CLANG

#if DOCTEST_GCC
#define DOCTEST_PRAGMA_TO_STR(x) _Pragma(#x)
#define DOCTEST_GCC_SUPPRESS_WARNING_PUSH _Pragma("GCC diagnostic push")
#define DOCTEST_GCC_SUPPRESS_WARNING(w) DOCTEST_PRAGMA_TO_STR(GCC diagnostic ignored w)
#define DOCTEST_GCC_SUPPRESS_WARNING_POP _Pragma("GCC diagnostic pop")
#define DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH(w)                                                  \
    DOCTEST_GCC_SUPPRESS_WARNING_PUSH DOCTEST_GCC_SUPPRESS_WARNING(w)
#else // DOCTEST_GCC
#define DOCTEST_GCC_SUPPRESS_WARNING_PUSH
#define DOCTEST_GCC_SUPPRESS_WARNING(w)
#define DOCTEST_GCC_SUPPRESS_WARNING_POP
#define DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH(w)
#endif // DOCTEST_GCC

#if DOCTEST_MSVC
#define DOCTEST_MSVC_SUPPRESS_WARNING_PUSH __pragma(warning(push))
#define DOCTEST_MSVC_SUPPRESS_WARNING(w) __pragma(warning(disable : w))
#define DOCTEST_MSVC_SUPPRESS_WARNING_POP __pragma(warning(pop))
#define DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(w)                                                 \
    DOCTEST_MSVC_SUPPRESS_WARNING_PUSH DOCTEST_MSVC_SUPPRESS_WARNING(w)
#else // DOCTEST_MSVC
#define DOCTEST_MSVC_SUPPRESS_WARNING_PUSH
#define DOCTEST_MSVC_SUPPRESS_WARNING(w)
#define DOCTEST_MSVC_SUPPRESS_WARNING_POP
#define DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(w)
#endif // DOCTEST_MSVC

// =================================================================================================
// == COMPILER WARNINGS ============================================================================
// =================================================================================================

// both the header and the implementation suppress all of these,
// so it only makes sense to aggregate them like so
#define DOCTEST_SUPPRESS_COMMON_WARNINGS_PUSH                                                      \
    DOCTEST_CLANG_SUPPRESS_WARNING_PUSH                                                            \
    DOCTEST_CLANG_SUPPRESS_WARNING("-Wunknown-pragmas")                                            \
    DOCTEST_CLANG_SUPPRESS_WARNING("-Wweak-vtables")                                               \
    DOCTEST_CLANG_SUPPRESS_WARNING("-Wpadded")                                                     \
    DOCTEST_CLANG_SUPPRESS_WARNING("-Wmissing-prototypes")                                         \
    DOCTEST_CLANG_SUPPRESS_WARNING("-Wc++98-compat")                                               \
    DOCTEST_CLANG_SUPPRESS_WARNING("-Wc++98-compat-pedantic")                                      \
                                                                                                   \
    DOCTEST_GCC_SUPPRESS_WARNING_PUSH                                                              \
    DOCTEST_GCC_SUPPRESS_WARNING("-Wunknown-pragmas")                                              \
    DOCTEST_GCC_SUPPRESS_WARNING("-Wpragmas")                                                      \
    DOCTEST_GCC_SUPPRESS_WARNING("-Weffc++")                                                       \
    DOCTEST_GCC_SUPPRESS_WARNING("-Wstrict-overflow")                                              \
    DOCTEST_GCC_SUPPRESS_WARNING("-Wstrict-aliasing")                                              \
    DOCTEST_GCC_SUPPRESS_WARNING("-Wmissing-declarations")                                         \
    DOCTEST_GCC_SUPPRESS_WARNING("-Wuseless-cast")                                                 \
    DOCTEST_GCC_SUPPRESS_WARNING("-Wnoexcept")                                                     \
                                                                                                   \
    DOCTEST_MSVC_SUPPRESS_WARNING_PUSH                                                             \
    /* these 4 also disabled globally via cmake: */                                                \
    DOCTEST_MSVC_SUPPRESS_WARNING(4514) /* unreferenced inline function has been removed */        \
    DOCTEST_MSVC_SUPPRESS_WARNING(4571) /* SEH related */                                          \
    DOCTEST_MSVC_SUPPRESS_WARNING(4710) /* function not inlined */                                 \
    DOCTEST_MSVC_SUPPRESS_WARNING(4711) /* function selected for inline expansion*/                \
    /* common ones */                                                                              \
    DOCTEST_MSVC_SUPPRESS_WARNING(4616) /* invalid compiler warning */                             \
    DOCTEST_MSVC_SUPPRESS_WARNING(4619) /* invalid compiler warning */                             \
    DOCTEST_MSVC_SUPPRESS_WARNING(4996) /* The compiler encountered a deprecated declaration */    \
    DOCTEST_MSVC_SUPPRESS_WARNING(4706) /* assignment within conditional expression */             \
    DOCTEST_MSVC_SUPPRESS_WARNING(4512) /* 'class' : assignment operator could not be generated */ \
    DOCTEST_MSVC_SUPPRESS_WARNING(4127) /* conditional expression is constant */                   \
    DOCTEST_MSVC_SUPPRESS_WARNING(4820) /* padding */                                              \
    DOCTEST_MSVC_SUPPRESS_WARNING(4625) /* copy constructor was implicitly deleted */              \
    DOCTEST_MSVC_SUPPRESS_WARNING(4626) /* assignment operator was implicitly deleted */           \
    DOCTEST_MSVC_SUPPRESS_WARNING(5027) /* move assignment operator implicitly deleted */          \
    DOCTEST_MSVC_SUPPRESS_WARNING(5026) /* move constructor was implicitly deleted */              \
    DOCTEST_MSVC_SUPPRESS_WARNING(4640) /* construction of local static object not thread-safe */  \
    DOCTEST_MSVC_SUPPRESS_WARNING(5045) /* Spectre mitigation for memory load */                   \
    DOCTEST_MSVC_SUPPRESS_WARNING(5264) /* 'variable-name': 'const' variable is not used */        \
    /* static analysis */                                                                          \
    DOCTEST_MSVC_SUPPRESS_WARNING(26439) /* Function may not throw. Declare it 'noexcept' */       \
    DOCTEST_MSVC_SUPPRESS_WARNING(26495) /* Always initialize a member variable */                 \
    DOCTEST_MSVC_SUPPRESS_WARNING(26451) /* Arithmetic overflow ... */                             \
    DOCTEST_MSVC_SUPPRESS_WARNING(26444) /* Avoid unnamed objects with custom ctor and dtor... */  \
    DOCTEST_MSVC_SUPPRESS_WARNING(26812) /* Prefer 'enum class' over 'enum' */

#define DOCTEST_SUPPRESS_COMMON_WARNINGS_POP                                                       \
    DOCTEST_CLANG_SUPPRESS_WARNING_POP                                                             \
    DOCTEST_GCC_SUPPRESS_WARNING_POP                                                               \
    DOCTEST_MSVC_SUPPRESS_WARNING_POP

DOCTEST_SUPPRESS_COMMON_WARNINGS_PUSH

DOCTEST_CLANG_SUPPRESS_WARNING_PUSH
DOCTEST_CLANG_SUPPRESS_WARNING("-Wnon-virtual-dtor")
DOCTEST_CLANG_SUPPRESS_WARNING("-Wdeprecated")

DOCTEST_GCC_SUPPRESS_WARNING_PUSH
DOCTEST_GCC_SUPPRESS_WARNING("-Wctor-dtor-privacy")
DOCTEST_GCC_SUPPRESS_WARNING("-Wnon-virtual-dtor")
DOCTEST_GCC_SUPPRESS_WARNING("-Wsign-promo")

DOCTEST_MSVC_SUPPRESS_WARNING_PUSH
DOCTEST_MSVC_SUPPRESS_WARNING(4623) // default constructor was implicitly defined as deleted

#define DOCTEST_MAKE_STD_HEADERS_CLEAN_FROM_WARNINGS_ON_WALL_BEGIN                                 \
    DOCTEST_MSVC_SUPPRESS_WARNING_PUSH                                                             \
    DOCTEST_MSVC_SUPPRESS_WARNING(4548) /* before comma no effect; expected side - effect */       \
    DOCTEST_MSVC_SUPPRESS_WARNING(4265) /* virtual functions, but destructor is not virtual */     \
    DOCTEST_MSVC_SUPPRESS_WARNING(4986) /* exception specification does not match previous */      \
    DOCTEST_MSVC_SUPPRESS_WARNING(4350) /* 'member1' called instead of 'member2' */                \
    DOCTEST_MSVC_SUPPRESS_WARNING(4668) /* not defined as a preprocessor macro */                  \
    DOCTEST_MSVC_SUPPRESS_WARNING(4365) /* signed/unsigned mismatch */                             \
    DOCTEST_MSVC_SUPPRESS_WARNING(4774) /* format string not a string literal */                   \
    DOCTEST_MSVC_SUPPRESS_WARNING(4820) /* padding */                                              \
    DOCTEST_MSVC_SUPPRESS_WARNING(4625) /* copy constructor was implicitly deleted */              \
    DOCTEST_MSVC_SUPPRESS_WARNING(4626) /* assignment operator was implicitly deleted */           \
    DOCTEST_MSVC_SUPPRESS_WARNING(5027) /* move assignment operator implicitly deleted */          \
    DOCTEST_MSVC_SUPPRESS_WARNING(5026) /* move constructor was implicitly deleted */              \
    DOCTEST_MSVC_SUPPRESS_WARNING(4623) /* default constructor was implicitly deleted */           \
    DOCTEST_MSVC_SUPPRESS_WARNING(5039) /* pointer to pot. throwing function passed to extern C */ \
    DOCTEST_MSVC_SUPPRESS_WARNING(5045) /* Spectre mitigation for memory load */                   \
    DOCTEST_MSVC_SUPPRESS_WARNING(5105) /* macro producing 'defined' has undefined behavior */     \
    DOCTEST_MSVC_SUPPRESS_WARNING(4738) /* storing float result in memory, loss of performance */  \
    DOCTEST_MSVC_SUPPRESS_WARNING(5262) /* implicit fall-through */

#define DOCTEST_MAKE_STD_HEADERS_CLEAN_FROM_WARNINGS_ON_WALL_END DOCTEST_MSVC_SUPPRESS_WARNING_POP

// =================================================================================================
// == FEATURE DETECTION ============================================================================
// =================================================================================================

// general compiler feature support table: https://en.cppreference.com/w/cpp/compiler_support
// MSVC C++11 feature support table: https://msdn.microsoft.com/en-us/library/hh567368.aspx
// GCC C++11 feature support table: https://gcc.gnu.org/projects/cxx-status.html
// MSVC version table:
// https://en.wikipedia.org/wiki/Microsoft_Visual_C%2B%2B#Internal_version_numbering
// MSVC++ 14.3 (17) _MSC_VER == 1930 (Visual Studio 2022)
// MSVC++ 14.2 (16) _MSC_VER == 1920 (Visual Studio 2019)
// MSVC++ 14.1 (15) _MSC_VER == 1910 (Visual Studio 2017)
// MSVC++ 14.0      _MSC_VER == 1900 (Visual Studio 2015)
// MSVC++ 12.0      _MSC_VER == 1800 (Visual Studio 2013)
// MSVC++ 11.0      _MSC_VER == 1700 (Visual Studio 2012)
// MSVC++ 10.0      _MSC_VER == 1600 (Visual Studio 2010)
// MSVC++ 9.0       _MSC_VER == 1500 (Visual Studio 2008)
// MSVC++ 8.0       _MSC_VER == 1400 (Visual Studio 2005)

// Universal Windows Platform support
#if defined(WINAPI_FAMILY) && (WINAPI_FAMILY == WINAPI_FAMILY_APP)
#define DOCTEST_CONFIG_NO_WINDOWS_SEH
#endif // WINAPI_FAMILY
#if DOCTEST_MSVC && !defined(DOCTEST_CONFIG_WINDOWS_SEH)
#define DOCTEST_CONFIG_WINDOWS_SEH
#endif // MSVC
#if defined(DOCTEST_CONFIG_NO_WINDOWS_SEH) && defined(DOCTEST_CONFIG_WINDOWS_SEH)
#undef DOCTEST_CONFIG_WINDOWS_SEH
#endif // DOCTEST_CONFIG_NO_WINDOWS_SEH

#if !defined(_WIN32) && !defined(__QNX__) && !defined(DOCTEST_CONFIG_POSIX_SIGNALS) &&             \
        !defined(__EMSCRIPTEN__) && !defined(__wasi__)
#define DOCTEST_CONFIG_POSIX_SIGNALS
#endif // _WIN32
#if defined(DOCTEST_CONFIG_NO_POSIX_SIGNALS) && defined(DOCTEST_CONFIG_POSIX_SIGNALS)
#undef DOCTEST_CONFIG_POSIX_SIGNALS
#endif // DOCTEST_CONFIG_NO_POSIX_SIGNALS

```

```
#ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
#if !defined(__cpp_exceptions) && !defined(__EXCEPTIONS) && !defined(_CPPUNWIND)                   \
        || defined(__wasi__)
#define DOCTEST_CONFIG_NO_EXCEPTIONS
#endif // no exceptions
#endif // DOCTEST_CONFIG_NO_EXCEPTIONS

#ifdef DOCTEST_CONFIG_NO_EXCEPTIONS_BUT_WITH_ALL_ASSERTS
#ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
#define DOCTEST_CONFIG_NO_EXCEPTIONS
#endif // DOCTEST_CONFIG_NO_EXCEPTIONS
#endif // DOCTEST_CONFIG_NO_EXCEPTIONS_BUT_WITH_ALL_ASSERTS

#if defined(DOCTEST_CONFIG_NO_EXCEPTIONS) && !defined(DOCTEST_CONFIG_NO_TRY_CATCH_IN_ASSERTS)
#define DOCTEST_CONFIG_NO_TRY_CATCH_IN_ASSERTS
#endif // DOCTEST_CONFIG_NO_EXCEPTIONS && !DOCTEST_CONFIG_NO_TRY_CATCH_IN_ASSERTS

#ifdef __wasi__
#define DOCTEST_CONFIG_NO_MULTITHREADING
#endif

#if defined(DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN) && !defined(DOCTEST_CONFIG_IMPLEMENT)
#define DOCTEST_CONFIG_IMPLEMENT
#endif // DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

#if defined(_WIN32) || defined(__CYGWIN__)
#if DOCTEST_MSVC
#define DOCTEST_SYMBOL_EXPORT __declspec(dllexport)
#define DOCTEST_SYMBOL_IMPORT __declspec(dllimport)
#else // MSVC
#define DOCTEST_SYMBOL_EXPORT __attribute__((dllexport))
#define DOCTEST_SYMBOL_IMPORT __attribute__((dllimport))
#endif // MSVC
#else  // _WIN32
#define DOCTEST_SYMBOL_EXPORT __attribute__((visibility("default")))
#define DOCTEST_SYMBOL_IMPORT
#endif // _WIN32

#ifdef DOCTEST_CONFIG_IMPLEMENTATION_IN_DLL
#ifdef DOCTEST_CONFIG_IMPLEMENT
#define DOCTEST_INTERFACE DOCTEST_SYMBOL_EXPORT
#else // DOCTEST_CONFIG_IMPLEMENT
#define DOCTEST_INTERFACE DOCTEST_SYMBOL_IMPORT
#endif // DOCTEST_CONFIG_IMPLEMENT
#else  // DOCTEST_CONFIG_IMPLEMENTATION_IN_DLL
#define DOCTEST_INTERFACE
#endif // DOCTEST_CONFIG_IMPLEMENTATION_IN_DLL

// needed for extern template instantiations
// see https://github.com/fmtlib/fmt/issues/2228
#if DOCTEST_MSVC
#define DOCTEST_INTERFACE_DECL
#define DOCTEST_INTERFACE_DEF DOCTEST_INTERFACE
#else // DOCTEST_MSVC
#define DOCTEST_INTERFACE_DECL DOCTEST_INTERFACE
#define DOCTEST_INTERFACE_DEF
#endif // DOCTEST_MSVC

#define DOCTEST_EMPTY

#if DOCTEST_MSVC
#define DOCTEST_NOINLINE __declspec(noinline)
#define DOCTEST_UNUSED
#define DOCTEST_ALIGNMENT(x)
#elif DOCTEST_CLANG && DOCTEST_CLANG < DOCTEST_COMPILER(3, 5, 0)
#define DOCTEST_NOINLINE
#define DOCTEST_UNUSED
#define DOCTEST_ALIGNMENT(x)
#else
#define DOCTEST_NOINLINE __attribute__((noinline))
#define DOCTEST_UNUSED __attribute__((unused))
#define DOCTEST_ALIGNMENT(x) __attribute__((aligned(x)))
#endif

#ifdef DOCTEST_CONFIG_NO_CONTRADICTING_INLINE
#define DOCTEST_INLINE_NOINLINE inline
#else
#define DOCTEST_INLINE_NOINLINE inline DOCTEST_NOINLINE
#endif

#ifndef DOCTEST_NORETURN
#if DOCTEST_MSVC && (DOCTEST_MSVC < DOCTEST_COMPILER(19, 0, 0))
#define DOCTEST_NORETURN
#else // DOCTEST_MSVC
#define DOCTEST_NORETURN [[noreturn]]
#endif // DOCTEST_MSVC
#endif // DOCTEST_NORETURN
```

```

#ifndef DOCTEST_NOEXCEPT
#if DOCTEST_MSVC && (DOCTEST_MSVC < DOCTEST_COMPILER(19, 0, 0))
#define DOCTEST_NOEXCEPT
#else // DOCTEST_MSVC
#define DOCTEST_NOEXCEPT noexcept
#endif // DOCTEST_MSVC
#endif // DOCTEST_NOEXCEPT

#ifndef DOCTEST_CONSTEXPR
#if DOCTEST_MSVC && (DOCTEST_MSVC < DOCTEST_COMPILER(19, 0, 0))
#define DOCTEST_CONSTEXPR const
#define DOCTEST_CONSTEXPR_FUNC inline
#else // DOCTEST_MSVC
#define DOCTEST_CONSTEXPR constexpr
#define DOCTEST_CONSTEXPR_FUNC constexpr
#endif // DOCTEST_MSVC
#endif // DOCTEST_CONSTEXPR

#ifndef DOCTEST_NO_SANITIZE_INTEGER
#if DOCTEST_CLANG >= DOCTEST_COMPILER(3, 7, 0)
#define DOCTEST_NO_SANITIZE_INTEGER __attribute__((no_sanitize("integer")))
#else
#define DOCTEST_NO_SANITIZE_INTEGER
#endif
#endif // DOCTEST_NO_SANITIZE_INTEGER

// =================================================================================================
// == FEATURE DETECTION END ========================================================================
// =================================================================================================

#define DOCTEST_DECLARE_INTERFACE(name)                                                            \
    virtual ~name();                                                                               \
    name() = default;                                                                              \
    name(const name&) = delete;                                                                    \
    name(name&&) = delete;                                                                         \
    name& operator=(const name&) = delete;                                                         \
    name& operator=(name&&) = delete;

#define DOCTEST_DEFINE_INTERFACE(name)                                                             \
    name::~name() = default;

// internal macros for string concatenation and anonymous variable name generation
#define DOCTEST_CAT_IMPL(s1, s2) s1##s2
#define DOCTEST_CAT(s1, s2) DOCTEST_CAT_IMPL(s1, s2)
#ifdef __COUNTER__ // not standard and may be missing for some compilers
#define DOCTEST_ANONYMOUS(x) DOCTEST_CAT(x, __COUNTER__)
#else // __COUNTER__
#define DOCTEST_ANONYMOUS(x) DOCTEST_CAT(x, __LINE__)
#endif // __COUNTER__

#ifndef DOCTEST_CONFIG_ASSERTION_PARAMETERS_BY_VALUE
#define DOCTEST_REF_WRAP(x) x&
#else // DOCTEST_CONFIG_ASSERTION_PARAMETERS_BY_VALUE
#define DOCTEST_REF_WRAP(x) x
#endif // DOCTEST_CONFIG_ASSERTION_PARAMETERS_BY_VALUE

// not using __APPLE__ because... this is how Catch does it
#ifdef __MAC_OS_X_VERSION_MIN_REQUIRED
#define DOCTEST_PLATFORM_MAC
#elif defined(__IPHONE_OS_VERSION_MIN_REQUIRED)
#define DOCTEST_PLATFORM_IPHONE
#elif defined(_WIN32)
#define DOCTEST_PLATFORM_WINDOWS
#elif defined(__wasi__)
#define DOCTEST_PLATFORM_WASI
#else // DOCTEST_PLATFORM
#define DOCTEST_PLATFORM_LINUX
#endif // DOCTEST_PLATFORM

namespace doctest { namespace detail {
    static DOCTEST_CONSTEXPR int consume(const int*, int) noexcept { return 0; }
}}

#define DOCTEST_GLOBAL_NO_WARNINGS(var, ...)                                                        \
    DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wglobal-constructors")                               \
    static const int var = doctest::detail::consume(&var, __VA_ARGS__);                            \
    DOCTEST_CLANG_SUPPRESS_WARNING_POP

#ifndef DOCTEST_BREAK_INTO_DEBUGGER
// should probably take a look at https://github.com/scottt/debugbreak
#ifdef DOCTEST_PLATFORM_LINUX
#if defined(__GNUC__) && (defined(__i386) || defined(__x86_64))
// Break at the location of the failing check if possible
#define DOCTEST_BREAK_INTO_DEBUGGER() __asm__("int $3\n" : :) // NOLINT(hicpp-no-assembler)
#else
#include <signal.h>
```

```
#define DOCTEST_BREAK_INTO_DEBUGGER() raise(SIGTRAP)
#endif
#elif defined(DOCTEST_PLATFORM_MAC)
#if defined(__x86_64) || defined(__x86_64__) || defined(__amd64__) || defined(__i386)
#define DOCTEST_BREAK_INTO_DEBUGGER() __asm__("int $3\n" : :) // NOLINT(hicpp-no-assembler)
#elif defined(__ppc__) || defined(__ppc64__)
// https://www.cocoawithlove.com/2008/03/break-into-debugger.html
#define DOCTEST_BREAK_INTO_DEBUGGER() __asm__("li r0, 20\nsc\nnop\nli r0, 37\nli r4, 2\nsc\nnop\n": : : "memory","r0","r3","r4") // NOLINT(hicpp-no-assembler)
#else
#define DOCTEST_BREAK_INTO_DEBUGGER() __asm__("brk #0"); // NOLINT(hicpp-no-assembler)
#endif
#elif DOCTEST_MSVC
#define DOCTEST_BREAK_INTO_DEBUGGER() __debugbreak()
#elif defined(__MINGW32__)
DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH("-Wredundant-decls")
extern "C" __declspec(dllimport) void __stdcall DebugBreak();
DOCTEST_GCC_SUPPRESS_WARNING_POP
#define DOCTEST_BREAK_INTO_DEBUGGER() ::DebugBreak()
#else // linux
#define DOCTEST_BREAK_INTO_DEBUGGER() (static_cast<void>(0))
#endif // linux
#endif // DOCTEST_BREAK_INTO_DEBUGGER

// this is kept here for backwards compatibility since the config option was changed
#ifdef DOCTEST_CONFIG_USE_IOSFWD
#ifndef DOCTEST_CONFIG_USE_STD_HEADERS
#define DOCTEST_CONFIG_USE_STD_HEADERS
#endif
#endif // DOCTEST_CONFIG_USE_IOSFWD

// for clang - always include ciso646 (which drags some std stuff) because
// we want to check if we are using libc++ with the _LIBCPP_VERSION macro in
// which case we don’t want to forward declare stuff from std - for reference:
// https://github.com/doctest/doctest/issues/126
// https://github.com/doctest/doctest/issues/356
#if DOCTEST_CLANG
#include <ciso646>
#endif // clang

#ifdef _LIBCPP_VERSION
#ifndef DOCTEST_CONFIG_USE_STD_HEADERS
#define DOCTEST_CONFIG_USE_STD_HEADERS
#endif
#endif // _LIBCPP_VERSION

#ifdef DOCTEST_CONFIG_USE_STD_HEADERS
#ifndef DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
#define DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
#endif // DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
DOCTEST_MAKE_STD_HEADERS_CLEAN_FROM_WARNINGS_ON_WALL_BEGIN
#include <cstddef>
#include <ostream>
#include <istream>
DOCTEST_MAKE_STD_HEADERS_CLEAN_FROM_WARNINGS_ON_WALL_END
#else // DOCTEST_CONFIG_USE_STD_HEADERS

// Forward declaring ’X’ in namespace std is not permitted by the C++ Standard.
DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4643)

namespace std { // NOLINT(cert-dcl58-cpp)
typedef decltype(nullptr) nullptr_t; // NOLINT(modernize-use-using)
typedef decltype(sizeof(void*)) size_t; // NOLINT(modernize-use-using)
template <class charT>
struct char_traits;
template <>
struct char_traits<char>;
template <class charT, class traits>
class basic_ostream; // NOLINT(fuchsia-virtual-inheritance)
typedef basic_ostream<char, char_traits<char>> ostream; // NOLINT(modernize-use-using)
template<class traits>
// NOLINTNEXTLINE
basic_ostream<char, traits>& operator«(basic_ostream<char, traits>&, const char*);
template <class charT, class traits>
class basic_istream;
typedef basic_istream<char, char_traits<char>> istream; // NOLINT(modernize-use-using)
template <class... Types>
class tuple;
#if DOCTEST_MSVC >= DOCTEST_COMPILER(19, 20, 0)
// see this issue on why this is needed: https://github.com/doctest/doctest/issues/183
template <class Ty>
class allocator;
template <class Elem, class Traits, class Alloc>
class basic_string;
using string = basic_string<char, char_traits<char>, allocator<char»;
#endif // VS 2019
} // namespace std
```

```

DOCTEST_MSVC_SUPPRESS_WARNING_POP

#endif // DOCTEST_CONFIG_USE_STD_HEADERS

#ifdef DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
#include <type_traits>
#endif // DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS

namespace doctest {

using std::size_t;

DOCTEST_INTERFACE extern bool is_running_in_test;

#ifndef DOCTEST_CONFIG_STRING_SIZE_TYPE
#define DOCTEST_CONFIG_STRING_SIZE_TYPE unsigned
#endif

// A 24 byte string class (can be as small as 17 for x64 and 13 for x86) that can hold strings with
length
// of up to 23 chars on the stack before going on the heap - the last byte of the buffer is used for:
// - "is small" bit - the highest bit - if "0" then it is small - otherwise its "1" (128)
// - if small - capacity left before going on the heap - using the lowest 5 bits
// - if small - 2 bits are left unused - the second and third highest ones
// - if small - acts as a null terminator if strlen() is 23 (24 including the null terminator)
//              and the "is small" bit remains "0" ("as well as the capacity left") so its OK
// Idea taken from this lecture about the string implementation of facebook/folly - fbstring
// https://www.youtube.com/watch?v=kPR8h4-qZdk
// TODO:
// - optimizations - like not deleting memory unnecessarily in operator= and etc.
// - resize/reserve/clear
// - replace
// - back/front
// - iterator stuff
// - find & friends
// - push_back/pop_back
// - assign/insert/erase
// - relational operators as free functions - taking const char* as one of the params
class DOCTEST_INTERFACE String
{
public:
    using size_type = DOCTEST_CONFIG_STRING_SIZE_TYPE;

private:
    static DOCTEST_CONSTEXPR size_type len  = 24;
    static DOCTEST_CONSTEXPR size_type last = len - 1;

    struct view // len should be more than sizeof(view) - because of the final byte for flags
    {
        char*     ptr;
        size_type size;
        size_type capacity;
    };

    union
    {
        char buf[len]; // NOLINT(*-avoid-c-arrays)
        view data;
    };

    char* allocate(size_type sz);

    bool isOnStack() const noexcept { return (buf[last] & 128) == 0; }
    void setOnHeap() noexcept;
    void setLast(size_type in = last) noexcept;
    void setSize(size_type sz) noexcept;

    void copy(const String& other);

public:
    static DOCTEST_CONSTEXPR size_type npos = static_cast<size_type>(-1);

    String() noexcept;
    ~String();

    // cppcheck-suppress noExplicitConstructor
    String(const char* in);
    String(const char* in, size_type in_size);

    String(std::istream& in, size_type in_size);

    String(const String& other);
    String& operator=(const String& other);

    String& operator+=(const String& other);

    String(String&& other) noexcept;
    String& operator=(String&& other) noexcept;

    char  operator[](size_type i) const;
    char& operator[](size_type i);

    // the only functions I'm willing to leave in the interface - available for inlining
    const char* c_str() const { return const_cast<String*>(this)->c_str(); } // NOLINT
    char*       c_str() {
        if (isOnStack()) {
            return reinterpret_cast<char*>(buf);
        }
        return data.ptr;
    }

    size_type size() const;
    size_type capacity() const;

    String substr(size_type pos, size_type cnt = npos) &&;
    String substr(size_type pos, size_type cnt = npos) const &;

    size_type find(char ch, size_type pos = 0) const;
    size_type rfind(char ch, size_type pos = npos) const;

    int compare(const char* other, bool no_case = false) const;
    int compare(const String& other, bool no_case = false) const;

friend DOCTEST_INTERFACE std::ostream& operator«(std::ostream& s, const String& in);
};

DOCTEST_INTERFACE String operator+(const String& lhs, const String& rhs);

DOCTEST_INTERFACE bool operator==(const String& lhs, const String& rhs);
DOCTEST_INTERFACE bool operator!=(const String& lhs, const String& rhs);
DOCTEST_INTERFACE bool operator<(const String& lhs, const String& rhs);
DOCTEST_INTERFACE bool operator>(const String& lhs, const String& rhs);
DOCTEST_INTERFACE bool operator<=(const String& lhs, const String& rhs);
DOCTEST_INTERFACE bool operator>=(const String& lhs, const String& rhs);

class DOCTEST_INTERFACE Contains {
public:
    explicit Contains(const String& string);

    bool checkWith(const String& other) const;

    String string;
};

DOCTEST_INTERFACE String toString(const Contains& in);

DOCTEST_INTERFACE bool operator==(const String& lhs, const Contains& rhs);
DOCTEST_INTERFACE bool operator==(const Contains& lhs, const String& rhs);
DOCTEST_INTERFACE bool operator!=(const String& lhs, const Contains& rhs);
DOCTEST_INTERFACE bool operator!=(const Contains& lhs, const String& rhs);

namespace Color {
    enum Enum
    {
        None = 0,
        White,
        Red,
        Green,
        Blue,
        Cyan,
        Yellow,
        Grey,

        Bright = 0x10,

        BrightRed   = Bright | Red,
        BrightGreen = Bright | Green,
        LightGrey   = Bright | Grey,
        BrightWhite = Bright | White
    };

    DOCTEST_INTERFACE std::ostream& operator«(std::ostream& s, Color::Enum code);
} // namespace Color

namespace assertType {
    enum Enum
    {
        // macro traits

        is_warn    = 1,
        is_check   = 2 * is_warn,
        is_require = 2 * is_check,

        is_normal      = 2 * is_require,
        is_throws      = 2 * is_normal,
        is_throws_as   = 2 * is_throws,
        is_throws_with = 2 * is_throws_as,
        is_nothrow     = 2 * is_throws_with,

        is_false = 2 * is_nothrow,
        is_unary = 2 * is_false, // not checked anywhere - used just to distinguish the types

        is_eq = 2 * is_unary,
        is_ne = 2 * is_eq,

        is_lt = 2 * is_ne,
        is_gt = 2 * is_lt,

        is_ge = 2 * is_gt,
        is_le = 2 * is_ge,

        // macro types

        DT_WARN    = is_normal | is_warn,
        DT_CHECK   = is_normal | is_check,
        DT_REQUIRE = is_normal | is_require,

        DT_WARN_FALSE    = is_normal | is_false | is_warn,
        DT_CHECK_FALSE   = is_normal | is_false | is_check,
        DT_REQUIRE_FALSE = is_normal | is_false | is_require,

        DT_WARN_THROWS    = is_throws | is_warn,
        DT_CHECK_THROWS   = is_throws | is_check,
        DT_REQUIRE_THROWS = is_throws | is_require,

        DT_WARN_THROWS_AS    = is_throws_as | is_warn,
        DT_CHECK_THROWS_AS   = is_throws_as | is_check,
        DT_REQUIRE_THROWS_AS = is_throws_as | is_require,

        DT_WARN_THROWS_WITH    = is_throws_with | is_warn,
        DT_CHECK_THROWS_WITH   = is_throws_with | is_check,
        DT_REQUIRE_THROWS_WITH = is_throws_with | is_require,

        DT_WARN_THROWS_WITH_AS    = is_throws_with | is_throws_as | is_warn,
        DT_CHECK_THROWS_WITH_AS   = is_throws_with | is_throws_as | is_check,
        DT_REQUIRE_THROWS_WITH_AS = is_throws_with | is_throws_as | is_require,

        DT_WARN_NOTHROW    = is_nothrow | is_warn,
        DT_CHECK_NOTHROW   = is_nothrow | is_check,
        DT_REQUIRE_NOTHROW = is_nothrow | is_require,

        DT_WARN_EQ    = is_normal | is_eq | is_warn,
        DT_CHECK_EQ   = is_normal | is_eq | is_check,
        DT_REQUIRE_EQ = is_normal | is_eq | is_require,

        DT_WARN_NE    = is_normal | is_ne | is_warn,
        DT_CHECK_NE   = is_normal | is_ne | is_check,
        DT_REQUIRE_NE = is_normal | is_ne | is_require,

        DT_WARN_GT    = is_normal | is_gt | is_warn,
        DT_CHECK_GT   = is_normal | is_gt | is_check,
        DT_REQUIRE_GT = is_normal | is_gt | is_require,

        DT_WARN_LT    = is_normal | is_lt | is_warn,
        DT_CHECK_LT   = is_normal | is_lt | is_check,
        DT_REQUIRE_LT = is_normal | is_lt | is_require,

        DT_WARN_GE    = is_normal | is_ge | is_warn,
        DT_CHECK_GE   = is_normal | is_ge | is_check,
        DT_REQUIRE_GE = is_normal | is_ge | is_require,

        DT_WARN_LE    = is_normal | is_le | is_warn,
        DT_CHECK_LE   = is_normal | is_le | is_check,
        DT_REQUIRE_LE = is_normal | is_le | is_require,

        DT_WARN_UNARY    = is_normal | is_unary | is_warn,
        DT_CHECK_UNARY   = is_normal | is_unary | is_check,
        DT_REQUIRE_UNARY = is_normal | is_unary | is_require,

        DT_WARN_UNARY_FALSE    = is_normal | is_false | is_unary | is_warn,
        DT_CHECK_UNARY_FALSE   = is_normal | is_false | is_unary | is_check,
        DT_REQUIRE_UNARY_FALSE = is_normal | is_false | is_unary | is_require,
    };
} // namespace assertType

DOCTEST_INTERFACE const char* assertString(assertType::Enum at);
DOCTEST_INTERFACE const char* failureString(assertType::Enum at);
DOCTEST_INTERFACE const char* skipPathFromFilename(const char* file);

struct DOCTEST_INTERFACE TestCaseData
```

```
{
    String      m_file;       // the file in which the test was registered (using String - see #350)
    unsigned    m_line;       // the line where the test was registered
    const char* m_name;       // name of the test case
    const char* m_test_suite; // the test suite in which the test was added
    const char* m_description;
    bool        m_skip;
    bool        m_no_breaks;
    bool        m_no_output;
    bool        m_may_fail;
    bool        m_should_fail;
    int         m_expected_failures;
    double      m_timeout;
};

struct DOCTEST_INTERFACE AssertData
{
    // common - for all asserts
    const TestCaseData* m_test_case;
    assertType::Enum    m_at;
    const char*         m_file;
    int                 m_line;
    const char*         m_expr;
    bool                m_failed;

    // exception-related - for all asserts
    bool   m_threw;
    String m_exception;

    // for normal asserts
    String m_decomp;

    // for specific exception-related asserts
    bool           m_threw_as;
    const char*    m_exception_type;

    class DOCTEST_INTERFACE StringContains {
        private:
            Contains content;
            bool isContains;

        public:
            StringContains(const String& str) : content(str), isContains(false) { }
            StringContains(Contains cntn) : content(static_cast<Contains&&>(cntn)), isContains(true) { }

            bool check(const String& str) { return isContains ? (content == str) : (content.string == str); }

            operator const String&() const { return content.string; }

            const char* c_str() const { return content.string.c_str(); }
    } m_exception_string;

    AssertData(assertType::Enum at, const char* file, int line, const char* expr,
        const char* exception_type, const StringContains& exception_string);
};

struct DOCTEST_INTERFACE MessageData
{
    String           m_string;
    const char*      m_file;
    int              m_line;
    assertType::Enum m_severity;
};

struct DOCTEST_INTERFACE SubcaseSignature
{
    String      m_name;
    const char* m_file;
    int         m_line;

    bool operator==(const SubcaseSignature& other) const;
    bool operator<(const SubcaseSignature& other) const;
};

struct DOCTEST_INTERFACE IContextScope
{
    DOCTEST_DECLARE_INTERFACE(IContextScope)
    virtual void stringify(std::ostream*) const = 0;
};

namespace detail {
    struct DOCTEST_INTERFACE TestCase;
} // namespace detail

struct ContextOptions
```

```
{
    std::ostream* cout = nullptr; // stdout stream
    String        binary_name;    // the test binary name

    const detail::TestCase* currentTest = nullptr;

    // == parameters from the command line
    String   out;       // output filename
    String   order_by;  // how tests should be ordered
    unsigned rand_seed; // the seed for rand ordering

    unsigned first; // the first (matching) test to be executed
    unsigned last;  // the last (matching) test to be executed

    int abort_after;           // stop tests after this many failed assertions
    int subcase_filter_levels; // apply the subcase filters for the first N levels

    bool success;              // include successful assertions in output
    bool case_sensitive;       // if filtering should be case sensitive
    bool exit;                 // if the program should be exited after the tests are ran/whatever
    bool duration;             // print the time duration of each test case
    bool minimal;              // minimal console output (only test failures)
    bool quiet;                // no console output
    bool no_throw;             // to skip exceptions-related assertion macros
    bool no_exitcode;          // if the framework should return 0 as the exitcode
    bool no_run;               // to not run the tests at all (can be done with an "*" exclude)
    bool no_intro;             // to not print the intro of the framework
    bool no_version;           // to not print the version of the framework
    bool no_colors;            // if output to the console should be colorized
    bool force_colors;         // forces the use of colors even when a tty cannot be detected
    bool no_breaks;            // to not break into the debugger
    bool no_skip;              // don't skip test cases which are marked to be skipped
    bool gnu_file_line;        // if line numbers should be surrounded with :x: and not (x):
    bool no_path_in_filenames; // if the path to files should be removed from the output
    String strip_file_prefixes;// remove the longest matching one of these prefixes from any file
paths in the output
    bool no_line_numbers;      // if source code line numbers should be omitted from the output
    bool no_debug_output;      // no output in the debug console when a debugger is attached
    bool no_skipped_summary;   // don't print "skipped" in the summary !!! UNDOCUMENTED !!!
    bool no_time_in_output;    // omit any time/timestamps from output !!! UNDOCUMENTED !!!

    bool help;             // to print the help
    bool version;          // to print the version
    bool count;            // if only the count of matching tests is to be retrieved
    bool list_test_cases;  // to list all tests matching the filters
    bool list_test_suites; // to list all suites matching the filters
    bool list_reporters;   // lists all registered reporters
};

namespace detail {
    namespace types {
#ifdef DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
        using namespace std;
#else
        template <bool COND, typename T = void>
        struct enable_if { };

        template <typename T>
        struct enable_if<true, T> { using type = T; };

        struct true_type { static DOCTEST_CONSTEXPR bool value = true; };
        struct false_type { static DOCTEST_CONSTEXPR bool value = false; };

        template <typename T> struct remove_reference { using type = T; };
        template <typename T> struct remove_reference<T&> { using type = T; };
        template <typename T> struct remove_reference<T&&> { using type = T; };

        template <typename T> struct is_rvalue_reference : false_type { };
        template <typename T> struct is_rvalue_reference<T&&> : true_type { };

        template<typename T> struct remove_const { using type = T; };
        template <typename T> struct remove_const<const T> { using type = T; };

        // Compiler intrinsics
        template <typename T> struct is_enum { static DOCTEST_CONSTEXPR bool value = __is_enum(T); };
        template <typename T> struct underlying_type { using type = __underlying_type(T); };

        template <typename T> struct is_pointer : false_type { };
        template <typename T> struct is_pointer<T*> : true_type { };

        template <typename T> struct is_array : false_type { };
        // NOLINTNEXTLINE(*-avoid-c-arrays)
        template <typename T, size_t SIZE> struct is_array<T[SIZE]> : true_type { };
#endif
    }

    // <utility>
    template <typename T>
    T&& declval();

    template <class T>
    DOCTEST_CONSTEXPR_FUNC T&& forward(typename types::remove_reference<T>::type& t) DOCTEST_NOEXCEPT
{
        return static_cast<T&&>(t);
    }

    template <class T>
    DOCTEST_CONSTEXPR_FUNC T&& forward(typename types::remove_reference<T>::type&& t) DOCTEST_NOEXCEPT
{
        return static_cast<T&&>(t);
    }

    template <typename T>
    struct deferred_false : types::false_type { };

// MSVS 2015 :(
#if !DOCTEST_CLANG && defined(_MSC_VER) && _MSC_VER <= 1900
    template <typename T, typename = void>
    struct has_global_insertion_operator : types::false_type { };

    template <typename T>
    struct has_global_insertion_operator<T, decltype(::operator«(declval<std::ostream&>(),
declval<const T&>()), void())> : types::true_type { };

    template <typename T, typename = void>
    struct has_insertion_operator { static DOCTEST_CONSTEXPR bool value =
has_global_insertion_operator<T>::value; };

    template <typename T, bool global>
    struct insert_hack;

    template <typename T>
    struct insert_hack<T, true> {
        static void insert(std::ostream& os, const T& t) { ::operator«(os, t); }
    };

    template <typename T>
    struct insert_hack<T, false> {
        static void insert(std::ostream& os, const T& t) { operator«(os, t); }
    };

    template <typename T>
    using insert_hack_t = insert_hack<T, has_global_insertion_operator<T>::value>;
#else
    template <typename T, typename = void>
    struct has_insertion_operator : types::false_type { };
#endif

    template <typename T>
    struct has_insertion_operator<T, decltype(operator«(declval<std::ostream&>(), declval<const
T&>()), void())> : types::true_type { };

    template <typename T>
    struct should_stringify_as_underlying_type {
        static DOCTEST_CONSTEXPR bool value = detail::types::is_enum<T>::value &&
!doctest::detail::has_insertion_operator<T>::value;
    };

    DOCTEST_INTERFACE std::ostream* tlssPush();
    DOCTEST_INTERFACE String tlssPop();

    template <bool C>
    struct StringMakerBase {
        template <typename T>
        static String convert(const DOCTEST_REF_WRAP(T)) {
#ifdef DOCTEST_CONFIG_REQUIRE_STRINGIFICATION_FOR_ALL_USED_TYPES
            static_assert(deferred_false<T>::value, "No stringification detected for type T. See
string conversion manual");
#endif
            return "{?}";
        }
    };

    template <typename T>
    struct filldata;

    template <typename T>
    void filloss(std::ostream* stream, const T& in) {
        filldata<T>::fill(stream, in);
    }

    template <typename T, size_t N>
    void filloss(std::ostream* stream, const T (&in)[N]) { // NOLINT(*-avoid-c-arrays)
        // T[N], T(&)[N], T(&&)[N] have same behaviour.
        // Hence remove reference.
        filloss<typename types::remove_reference<decltype(in)>::type>(stream, in);
    }

    template <typename T>
    String toStream(const T& in) {
        std::ostream* stream = tlssPush();
        filloss(stream, in);
        return tlssPop();
    }

    template <>
    struct StringMakerBase<true> {
        template <typename T>
        static String convert(const DOCTEST_REF_WRAP(T) in) {
            return toStream(in);
        }
    };
} // namespace detail

template <typename T>
struct StringMaker : public detail::StringMakerBase<
    detail::has_insertion_operator<T>::value || detail::types::is_pointer<T>::value ||
detail::types::is_array<T>::value>
{};

#ifndef DOCTEST_STRINGIFY
#ifdef DOCTEST_CONFIG_DOUBLE_STRINGIFY
#define DOCTEST_STRINGIFY(...) toString(toString(__VA_ARGS__))
#else
#define DOCTEST_STRINGIFY(...) toString(__VA_ARGS__)
#endif
#endif

template <typename T>
String toString() {
#if DOCTEST_CLANG == 0 && DOCTEST_GCC == 0 && DOCTEST_ICC == 0
    String ret = __FUNCSIG__; // class doctest::String __cdecl doctest::toString<TYPE>(void)
    String::size_type beginPos = ret.find('<');
    return ret.substr(beginPos + 1, ret.size() - beginPos -
static_cast<String::size_type>(sizeof(">(void)")));
#else
    String ret = __PRETTY_FUNCTION__; // doctest::String toString() [with T = TYPE]
    String::size_type begin = ret.find('=') + 2;
    return ret.substr(begin, ret.size() - begin - 1);
#endif
}

template <typename T, typename
detail::types::enable_if<!detail::should_stringify_as_underlying_type<T>::value, bool>::type = true>
String toString(const DOCTEST_REF_WRAP(T) value) {
    return StringMaker<T>::convert(value);
}

#ifdef DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
DOCTEST_INTERFACE String toString(const char* in);
#endif // DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING

#if DOCTEST_MSVC >= DOCTEST_COMPILER(19, 20, 0)
// see this issue on why this is needed: https://github.com/doctest/doctest/issues/183
DOCTEST_INTERFACE String toString(const std::string& in);
#endif // VS 2019

DOCTEST_INTERFACE String toString(String in);

DOCTEST_INTERFACE String toString(std::nullptr_t);

DOCTEST_INTERFACE String toString(bool in);

DOCTEST_INTERFACE String toString(float in);
DOCTEST_INTERFACE String toString(double in);
DOCTEST_INTERFACE String toString(double long in);

DOCTEST_INTERFACE String toString(char in);
DOCTEST_INTERFACE String toString(char signed in);
DOCTEST_INTERFACE String toString(char unsigned in);
DOCTEST_INTERFACE String toString(short in);
DOCTEST_INTERFACE String toString(short unsigned in);
DOCTEST_INTERFACE String toString(signed in);
DOCTEST_INTERFACE String toString(unsigned in);
DOCTEST_INTERFACE String toString(long in);
DOCTEST_INTERFACE String toString(long unsigned in);
DOCTEST_INTERFACE String toString(long long in);
DOCTEST_INTERFACE String toString(long long unsigned in);

template <typename T, typename
detail::types::enable_if<detail::should_stringify_as_underlying_type<T>::value, bool>::type = true>
```

```
String toString(const DOCTEST_REF_WRAP(T) value) {
    using UT = typename detail::types::underlying_type<T>::type;
    return (DOCTEST_STRINGIFY(static_cast<UT>(value)));
}

namespace detail {
    template <typename T>
    struct filldata
    {
        static void fill(std::ostream* stream, const T& in) {
#if defined(_MSC_VER) && _MSC_VER <= 1900
        insert_hack_t<T>::insert(*stream, in);
#else
        operator«(*stream, in);
#endif
        }
    };

DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4866)
// NOLINTBEGIN(*-avoid-c-arrays)
    template <typename T, size_t N>
    struct filldata<T[N]> {
        static void fill(std::ostream* stream, const T(&in)[N]) {
            *stream « "[";
            for (size_t i = 0; i < N; i++) {
                if (i != 0) { *stream « ", "; }
                *stream « (DOCTEST_STRINGIFY(in[i]));
            }
            *stream « "]";
        }
    };
// NOLINTEND(*-avoid-c-arrays)
DOCTEST_MSVC_SUPPRESS_WARNING_POP

    // Specialized since we don't want the terminating null byte!
// NOLINTBEGIN(*-avoid-c-arrays)
    template <size_t N>
    struct filldata<const char[N]> {
        static void fill(std::ostream* stream, const char (&in)[N]) {
            *stream « String(in, in[N - 1] ? N : N - 1);
        } // NOLINT(clang-analyzer-cplusplus.NewDeleteLeaks)
    };
// NOLINTEND(*-avoid-c-arrays)

    template <>
    struct filldata<const void*> {
        static void fill(std::ostream* stream, const void* in);
    };

    template <typename T>
    struct filldata<T*> {
DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4180)
        static void fill(std::ostream* stream, const T* in) {
DOCTEST_MSVC_SUPPRESS_WARNING_POP
DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wmicrosoft-cast")
            filldata<const void*>::fill(stream,
#if DOCTEST_GCC == 0 || DOCTEST_GCC >= DOCTEST_COMPILER(4, 9, 0)
                reinterpret_cast<const void*>(in)
#else
                *reinterpret_cast<const void* const*>(&in)
#endif
            );
DOCTEST_CLANG_SUPPRESS_WARNING_POP
        }
    };
}

struct DOCTEST_INTERFACE Approx
{
    Approx(double value);

    Approx operator()(double value) const;

#ifdef DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
    template <typename T>
    explicit Approx(const T& value,
                    typename detail::types::enable_if<std::is_constructible<double, T>::value>::type*
=
                            static_cast<T*>(nullptr)) {
        *this = static_cast<double>(value);
    }
#endif // DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS

    Approx& epsilon(double newEpsilon);

#ifdef DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
    template <typename T>
    typename std::enable_if<std::is_constructible<double, T>::value, Approx&>::type epsilon(
            const T& newEpsilon) {
        m_epsilon = static_cast<double>(newEpsilon);
        return *this;
    }
#endif //  DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS

    Approx& scale(double newScale);

#ifdef DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
    template <typename T>
    typename std::enable_if<std::is_constructible<double, T>::value, Approx&>::type scale(
            const T& newScale) {
        m_scale = static_cast<double>(newScale);
        return *this;
    }
#endif // DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS

    // clang-format off
    DOCTEST_INTERFACE friend bool operator==(double lhs, const Approx & rhs);
    DOCTEST_INTERFACE friend bool operator==(const Approx & lhs, double rhs);
    DOCTEST_INTERFACE friend bool operator!=(double lhs, const Approx & rhs);
    DOCTEST_INTERFACE friend bool operator!=(const Approx & lhs, double rhs);
    DOCTEST_INTERFACE friend bool operator<=(double lhs, const Approx & rhs);
    DOCTEST_INTERFACE friend bool operator<=(const Approx & lhs, double rhs);
    DOCTEST_INTERFACE friend bool operator>=(double lhs, const Approx & rhs);
    DOCTEST_INTERFACE friend bool operator>=(const Approx & lhs, double rhs);
    DOCTEST_INTERFACE friend bool operator< (double lhs, const Approx & rhs);
    DOCTEST_INTERFACE friend bool operator< (const Approx & lhs, double rhs);
    DOCTEST_INTERFACE friend bool operator> (double lhs, const Approx & rhs);
    DOCTEST_INTERFACE friend bool operator> (const Approx & lhs, double rhs);

#ifdef DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
#define DOCTEST_APPROX_PREFIX \
    template <typename T> friend typename std::enable_if<std::is_constructible<double, T>::value, bool>::type

    DOCTEST_APPROX_PREFIX operator==(const T& lhs, const Approx& rhs) { return operator==(static_cast<double>(lhs), rhs); }
    DOCTEST_APPROX_PREFIX operator==(const Approx& lhs, const T& rhs) { return operator==(rhs, lhs); }
    DOCTEST_APPROX_PREFIX operator!=(const T& lhs, const Approx& rhs) { return !operator==(lhs, rhs); }
    DOCTEST_APPROX_PREFIX operator!=(const Approx& lhs, const T& rhs) { return !operator==(rhs, lhs); }
    DOCTEST_APPROX_PREFIX operator<=(const T& lhs, const Approx& rhs) { return static_cast<double>(lhs) < rhs.m_value || lhs == rhs; }
    DOCTEST_APPROX_PREFIX operator<=(const Approx& lhs, const T& rhs) { return lhs.m_value < static_cast<double>(rhs) || lhs == rhs; }
    DOCTEST_APPROX_PREFIX operator>=(const T& lhs, const Approx& rhs) { return static_cast<double>(lhs) > rhs.m_value || lhs == rhs; }
    DOCTEST_APPROX_PREFIX operator>=(const Approx& lhs, const T& rhs) { return lhs.m_value > static_cast<double>(rhs) || lhs == rhs; }
    DOCTEST_APPROX_PREFIX operator< (const T& lhs, const Approx& rhs) { return static_cast<double>(lhs) < rhs.m_value && lhs != rhs; }
    DOCTEST_APPROX_PREFIX operator< (const Approx& lhs, const T& rhs) { return lhs.m_value < static_cast<double>(rhs) && lhs != rhs; }
    DOCTEST_APPROX_PREFIX operator> (const T& lhs, const Approx& rhs) { return static_cast<double>(lhs) > rhs.m_value && lhs != rhs; }
    DOCTEST_APPROX_PREFIX operator> (const Approx& lhs, const T& rhs) { return lhs.m_value > static_cast<double>(rhs) && lhs != rhs; }
#undef DOCTEST_APPROX_PREFIX
#endif // DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS

    // clang-format on

    double m_epsilon;
    double m_scale;
    double m_value;
};

DOCTEST_INTERFACE String toString(const Approx& in);

DOCTEST_INTERFACE const ContextOptions* getContextOptions();

template <typename F>
struct DOCTEST_INTERFACE_DECL IsNaN
{
    F value; bool flipped;
    IsNaN(F f, bool flip = false) : value(f), flipped(flip) { }
    IsNaN<F> operator!() const { return { value, !flipped }; }
    operator bool() const;
};
#ifndef __MINGW32__
extern template struct DOCTEST_INTERFACE_DECL IsNaN<float>;
extern template struct DOCTEST_INTERFACE_DECL IsNaN<double>;
extern template struct DOCTEST_INTERFACE_DECL IsNaN<long double>;
#endif
```

```
DOCTEST_INTERFACE String toString(IsNaN<float> in);
DOCTEST_INTERFACE String toString(IsNaN<double> in);
DOCTEST_INTERFACE String toString(IsNaN<double long> in);

#ifndef DOCTEST_CONFIG_DISABLE

namespace detail {
    // clang-format off
#ifdef DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
    template<class T>               struct decay_array       { using type = T; };
    template<class T, unsigned N>   struct decay_array<T[N]> { using type = T*; };
    template<class T>               struct decay_array<T[]>  { using type = T*; };

    template<class T>   struct not_char_pointer              { static DOCTEST_CONSTEXPR int value = 1;
};
    template<>          struct not_char_pointer<char*>       { static DOCTEST_CONSTEXPR int value = 0;
};
    template<>          struct not_char_pointer<const char*> { static DOCTEST_CONSTEXPR int value = 0;
};

    template<class T> struct can_use_op : public not_char_pointer<typename decay_array<T>::type> {};
#endif // DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
    // clang-format on

    struct DOCTEST_INTERFACE TestFailureException
    {
    };

    DOCTEST_INTERFACE bool checkIfShouldThrow(assertType::Enum at);

#ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
    DOCTEST_NORETURN
#endif // DOCTEST_CONFIG_NO_EXCEPTIONS
    DOCTEST_INTERFACE void throwException();

    struct DOCTEST_INTERFACE Subcase
    {
        SubcaseSignature m_signature;
        bool             m_entered = false;

        Subcase(const String& name, const char* file, int line);
        Subcase(const Subcase&) = delete;
        Subcase(Subcase&&) = delete;
        Subcase& operator=(const Subcase&) = delete;
        Subcase& operator=(Subcase&&) = delete;
        ~Subcase();

        operator bool() const;

        private:
            bool checkFilters();
    };

    template <typename L, typename R>
    String stringifyBinaryExpr(const DOCTEST_REF_WRAP(L) lhs, const char* op,
                               const DOCTEST_REF_WRAP(R) rhs) {
        return (DOCTEST_STRINGIFY(lhs)) + op + (DOCTEST_STRINGIFY(rhs));
    }

#if DOCTEST_CLANG && DOCTEST_CLANG < DOCTEST_COMPILER(3, 6, 0)
DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wunused-comparison")
#endif

// This will check if there is any way it could find a operator like member or friend and uses it.
// If not it doesn't find the operator or if the operator at global scope is defined after
// this template, the template won't be instantiated due to SFINAE. Once the template is not
// instantiated it can look for global operator using normal conversions.
#ifdef __NVCC__
#define SFINAE_OP(ret,op) ret
#else
#define SFINAE_OP(ret,op) decltype((void)(doctest::detail::declval<L>() op
doctest::detail::declval<R>()),ret{})
#endif

#define DOCTEST_DO_BINARY_EXPRESSION_COMPARISON(op, op_str, op_macro)                              \
    template <typename R>                                                                          \
    DOCTEST_NOINLINE SFINAE_OP(Result,op) operator op(R&& rhs) {                                   \
    bool res = op_macro(doctest::detail::forward<const L>(lhs), doctest::detail::forward<R>(rhs)); \
        if(m_at & assertType::is_false)                                                            \
            res = !res;                                                                            \
        if(!res || doctest::getContextOptions()->success)                                          \
            return Result(res, stringifyBinaryExpr(lhs, op_str, rhs));                             \
        return Result(res);                                                                        \
    }

    // more checks could be added - like in Catch:
    // https://github.com/catchorg/Catch2/pull/1480/files
    // https://github.com/catchorg/Catch2/pull/1481/files
#define DOCTEST_FORBIT_EXPRESSION(rt, op)                                                           \
    template <typename R>                                                                           \
    rt& operator op(const R&) {                                                                     \
        static_assert(deferred_false<R>::value,                                                     \
                      "Expression Too Complex Please Rewrite As Binary Comparison!");               \
        return *this;                                                                               \
    }

    struct DOCTEST_INTERFACE Result // NOLINT(*-member-init)
    {
        bool   m_passed;
        String m_decomp;

        Result() = default; // TODO: Why do we need this? (To remove NOLINT)
        Result(bool passed, const String& decomposition = String());

        // forbidding some expressions based on this table: https://en.cppreference.com/w/cpp/language/operator_precedence
        DOCTEST_FORBIT_EXPRESSION(Result, &)
        DOCTEST_FORBIT_EXPRESSION(Result, ^)
        DOCTEST_FORBIT_EXPRESSION(Result, |)
        DOCTEST_FORBIT_EXPRESSION(Result, &&)
        DOCTEST_FORBIT_EXPRESSION(Result, ||)
        DOCTEST_FORBIT_EXPRESSION(Result, ==)
        DOCTEST_FORBIT_EXPRESSION(Result, !=)
        DOCTEST_FORBIT_EXPRESSION(Result, <)
        DOCTEST_FORBIT_EXPRESSION(Result, >)
        DOCTEST_FORBIT_EXPRESSION(Result, <=)
        DOCTEST_FORBIT_EXPRESSION(Result, >=)
        DOCTEST_FORBIT_EXPRESSION(Result, =)
        DOCTEST_FORBIT_EXPRESSION(Result, +=)
        DOCTEST_FORBIT_EXPRESSION(Result, -=)
        DOCTEST_FORBIT_EXPRESSION(Result, *=)
        DOCTEST_FORBIT_EXPRESSION(Result, /=)
        DOCTEST_FORBIT_EXPRESSION(Result, %=)
        DOCTEST_FORBIT_EXPRESSION(Result, «=)
        DOCTEST_FORBIT_EXPRESSION(Result, »=)
        DOCTEST_FORBIT_EXPRESSION(Result, &=)
        DOCTEST_FORBIT_EXPRESSION(Result, ^=)
        DOCTEST_FORBIT_EXPRESSION(Result, |=)
    };

#ifndef DOCTEST_CONFIG_NO_COMPARISON_WARNING_SUPPRESSION

    DOCTEST_CLANG_SUPPRESS_WARNING_PUSH
    DOCTEST_CLANG_SUPPRESS_WARNING("-Wsign-conversion")
    DOCTEST_CLANG_SUPPRESS_WARNING("-Wsign-compare")
    //DOCTEST_CLANG_SUPPRESS_WARNING("-Wdouble-promotion")
    //DOCTEST_CLANG_SUPPRESS_WARNING("-Wconversion")
    //DOCTEST_CLANG_SUPPRESS_WARNING("-Wfloat-equal")

    DOCTEST_GCC_SUPPRESS_WARNING_PUSH
    DOCTEST_GCC_SUPPRESS_WARNING("-Wsign-conversion")
    DOCTEST_GCC_SUPPRESS_WARNING("-Wsign-compare")
    //DOCTEST_GCC_SUPPRESS_WARNING("-Wdouble-promotion")
    //DOCTEST_GCC_SUPPRESS_WARNING("-Wconversion")
    //DOCTEST_GCC_SUPPRESS_WARNING("-Wfloat-equal")

    DOCTEST_MSVC_SUPPRESS_WARNING_PUSH
    // https://stackoverflow.com/questions/39479163 what's the difference between 4018 and 4389
    DOCTEST_MSVC_SUPPRESS_WARNING(4388) // signed/unsigned mismatch
    DOCTEST_MSVC_SUPPRESS_WARNING(4389) // 'operator' : signed/unsigned mismatch
    DOCTEST_MSVC_SUPPRESS_WARNING(4018) // 'expression' : signed/unsigned mismatch
    //DOCTEST_MSVC_SUPPRESS_WARNING(4805) // 'operation' : unsafe mix of type 'type' and type 'type' in operation

#endif // DOCTEST_CONFIG_NO_COMPARISON_WARNING_SUPPRESSION

    // clang-format off
#ifndef DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
#define DOCTEST_COMPARISON_RETURN_TYPE bool
#else // DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
#define DOCTEST_COMPARISON_RETURN_TYPE typename types::enable_if<can_use_op<L>::value || can_use_op<R>::value, bool>::type
    inline bool eq(const char* lhs, const char* rhs) { return String(lhs) == String(rhs); }
    inline bool ne(const char* lhs, const char* rhs) { return String(lhs) != String(rhs); }
    inline bool lt(const char* lhs, const char* rhs) { return String(lhs) <  String(rhs); }
    inline bool gt(const char* lhs, const char* rhs) { return String(lhs) >  String(rhs); }
    inline bool le(const char* lhs, const char* rhs) { return String(lhs) <= String(rhs); }
    inline bool ge(const char* lhs, const char* rhs) { return String(lhs) >= String(rhs); }
#endif // DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
    // clang-format on

#define DOCTEST_RELATIONAL_OP(name, op)                                                             \
    template <typename L, typename R>                                                               \
    DOCTEST_COMPARISON_RETURN_TYPE name(const DOCTEST_REF_WRAP(L) lhs,                              \
                                        const DOCTEST_REF_WRAP(R) rhs) {                           \
        return lhs op rhs;                                                                         \
    }

    DOCTEST_RELATIONAL_OP(eq, ==)
    DOCTEST_RELATIONAL_OP(ne, !=)
    DOCTEST_RELATIONAL_OP(lt, <)
    DOCTEST_RELATIONAL_OP(gt, >)
    DOCTEST_RELATIONAL_OP(le, <=)
    DOCTEST_RELATIONAL_OP(ge, >=)

#ifndef DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
#define DOCTEST_CMP_EQ(l, r) l == r
#define DOCTEST_CMP_NE(l, r) l != r
#define DOCTEST_CMP_GT(l, r) l > r
#define DOCTEST_CMP_LT(l, r) l < r
#define DOCTEST_CMP_GE(l, r) l >= r
#define DOCTEST_CMP_LE(l, r) l <= r
#else // DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
#define DOCTEST_CMP_EQ(l, r) eq(l, r)
#define DOCTEST_CMP_NE(l, r) ne(l, r)
#define DOCTEST_CMP_GT(l, r) gt(l, r)
#define DOCTEST_CMP_LT(l, r) lt(l, r)
#define DOCTEST_CMP_GE(l, r) ge(l, r)
#define DOCTEST_CMP_LE(l, r) le(l, r)
#endif // DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING

    template <typename L>
    // cppcheck-suppress copyCtorAndEqOperator
    struct Expression_lhs
    {
        L                lhs;
        assertType::Enum m_at;

        explicit Expression_lhs(L&& in, assertType::Enum at)
                : lhs(static_cast<L&&>(in))
                , m_at(at) {}

        DOCTEST_NOINLINE operator Result() {
// this is needed only for MSVC 2015
DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4800) // 'int': forcing value to bool
            bool res = static_cast<bool>(lhs);
DOCTEST_MSVC_SUPPRESS_WARNING_POP
            if(m_at & assertType::is_false) {
                res = !res;
            }

            if(!res || getContextOptions()->success) {
                return { res, (DOCTEST_STRINGIFY(lhs)) };
            }
            return { res };
        }

        /* This is required for user-defined conversions from Expression_lhs to L */
        operator L() const { return lhs; }

        // clang-format off
        DOCTEST_DO_BINARY_EXPRESSION_COMPARISON(==, " == ", DOCTEST_CMP_EQ)
        DOCTEST_DO_BINARY_EXPRESSION_COMPARISON(!=, " != ", DOCTEST_CMP_NE)
        DOCTEST_DO_BINARY_EXPRESSION_COMPARISON(>,  " >  ", DOCTEST_CMP_GT)
        DOCTEST_DO_BINARY_EXPRESSION_COMPARISON(<,  " <  ", DOCTEST_CMP_LT)
        DOCTEST_DO_BINARY_EXPRESSION_COMPARISON(>=, " >= ", DOCTEST_CMP_GE)
        DOCTEST_DO_BINARY_EXPRESSION_COMPARISON(<=, " <= ", DOCTEST_CMP_LE)
        // clang-format on

        // forbidding some expressions based on this table: https://en.cppreference.com/w/cpp/language/operator_precedence
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, &)
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, ^)
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, |)
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, &&)
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, ||)
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, =)
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, +=)
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, -=)
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, *=)
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, /=)
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, %=)
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, «=)
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, »=)
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, &=)
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, ^=)
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, |=)
        // these 2 are unfortunate because they should be allowed - they have higher precedence over the comparisons, but the
        // ExpressionDecomposer class uses the left shift operator to capture the left operand of the binary expression...
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, «)
        DOCTEST_FORBIT_EXPRESSION(Expression_lhs, »)
    };

#ifndef DOCTEST_CONFIG_NO_COMPARISON_WARNING_SUPPRESSION

    DOCTEST_CLANG_SUPPRESS_WARNING_POP
    DOCTEST_MSVC_SUPPRESS_WARNING_POP
    DOCTEST_GCC_SUPPRESS_WARNING_POP

#endif // DOCTEST_CONFIG_NO_COMPARISON_WARNING_SUPPRESSION

#if DOCTEST_CLANG && DOCTEST_CLANG < DOCTEST_COMPILER(3, 6, 0)
DOCTEST_CLANG_SUPPRESS_WARNING_POP
#endif

    struct DOCTEST_INTERFACE ExpressionDecomposer
    {
        assertType::Enum m_at;

        ExpressionDecomposer(assertType::Enum at);

        // The right operator for capturing expressions is "<=" instead of "«" (based on the operator precedence table)
        // but then there will be warnings from GCC about "-Wparentheses" and since "_Pragma()" is problematic this will stay for now...
        // https://github.com/catchorg/Catch2/issues/870
        // https://github.com/catchorg/Catch2/issues/565
        template <typename L>
        Expression_lhs<const L&&> operator«(const L&& operand) { //bitfields bind to universal ref but not const rvalue ref
            return Expression_lhs<const L&&>(static_cast<const L&&>(operand), m_at);
        }

        template <typename L,typename types::enable_if<!doctest::detail::types::is_rvalue_reference<L>::value,void >::type* = nullptr>
        Expression_lhs<const L&> operator«(const L &operand) {
            return Expression_lhs<const L&>(operand, m_at);
        }
    };

    struct DOCTEST_INTERFACE TestSuite
    {
        const char* m_test_suite = nullptr;
        const char* m_description = nullptr;
        bool        m_skip = false;
        bool        m_no_breaks = false;
        bool        m_no_output = false;
        bool        m_may_fail = false;
        bool        m_should_fail = false;
        int         m_expected_failures = 0;
        double      m_timeout = 0;

        TestSuite& operator*(const char* in);

        template <typename T>
        TestSuite& operator*(const T& in) {
            in.fill(*this);
            return *this;
        }
    };

    using funcType = void (*)();

    struct DOCTEST_INTERFACE TestCase : public TestCaseData
    {
        funcType m_test; // a function pointer to the test case

        String m_type; // for templated test cases - gets appended to the real name
        int m_template_id; // an ID used to distinguish between the different versions of a templated test case
        String m_full_name; // contains the name (only for templated test cases!) + the template type

        TestCase(funcType test, const char* file, unsigned line, const TestSuite& test_suite,
                 const String& type = String(), int template_id = -1);

        TestCase(const TestCase& other);
        TestCase(TestCase&&) = delete;

        DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(26434) // hides a non-virtual function
        TestCase& operator=(const TestCase& other);
        DOCTEST_MSVC_SUPPRESS_WARNING_POP

        TestCase& operator=(TestCase&&) = delete;

        TestCase& operator*(const char* in);

        template <typename T>
        TestCase& operator*(const T& in) {
            in.fill(*this);
            return *this;
        }

        bool operator<(const TestCase& other) const;

        ~TestCase() = default;
    };

    // forward declarations of functions used by the macros
    DOCTEST_INTERFACE int  regTest(const TestCase& tc);
    DOCTEST_INTERFACE int  setTestSuite(const TestSuite& ts);
    DOCTEST_INTERFACE bool isDebuggerActive();

    template<typename T>
    int instantiationHelper(const T&) { return 0; }

    namespace binaryAssertComparison {
        enum Enum
        {
            eq = 0,
            ne,
            gt,
            lt,
            ge,
            le
        };
    } // namespace binaryAssertComparison

    // clang-format off
    template <int, class L, class R> struct RelationalComparator     { bool operator()(const
DOCTEST_REF_WRAP(L),     const DOCTEST_REF_WRAP(R)     ) const { return false;        } };

#define DOCTEST_BINARY_RELATIONAL_OP(n, op) \
    template <class L, class R> struct RelationalComparator<n, L, R> { bool operator()(const
DOCTEST_REF_WRAP(L) lhs, const DOCTEST_REF_WRAP(R) rhs) const { return op(lhs, rhs); } };
    // clang-format on

    DOCTEST_BINARY_RELATIONAL_OP(0, doctest::detail::eq)
    DOCTEST_BINARY_RELATIONAL_OP(1, doctest::detail::ne)
    DOCTEST_BINARY_RELATIONAL_OP(2, doctest::detail::gt)
    DOCTEST_BINARY_RELATIONAL_OP(3, doctest::detail::lt)
    DOCTEST_BINARY_RELATIONAL_OP(4, doctest::detail::ge)
    DOCTEST_BINARY_RELATIONAL_OP(5, doctest::detail::le)

    struct DOCTEST_INTERFACE ResultBuilder : public AssertData
    {
        ResultBuilder(assertType::Enum at, const char* file, int line, const char* expr,
                      const char* exception_type = "", const String& exception_string = "");

        ResultBuilder(assertType::Enum at, const char* file, int line, const char* expr,
                      const char* exception_type, const Contains& exception_string);

        void setResult(const Result& res);

        template <int comparison, typename L, typename R>
        DOCTEST_NOINLINE bool binary_assert(const DOCTEST_REF_WRAP(L) lhs,
                                            const DOCTEST_REF_WRAP(R) rhs) {
            m_failed = !RelationalComparator<comparison, L, R>()(lhs, rhs);
            if (m_failed || getContextOptions()->success) {
                m_decomp = stringifyBinaryExpr(lhs, ", ", rhs);
            }
            return !m_failed;
        }

        template <typename L>
        DOCTEST_NOINLINE bool unary_assert(const DOCTEST_REF_WRAP(L) val) {
            m_failed = !val;

            if (m_at & assertType::is_false) {
                m_failed = !m_failed;
            }

            if (m_failed || getContextOptions()->success) {
                m_decomp = (DOCTEST_STRINGIFY(val));
            }

            return !m_failed;
        }

        void translateException();

        bool log();
        void react() const;
    };

    namespace assertAction {
        enum Enum
        {
            nothing     = 0,
            dbgbreak    = 1,
            shouldthrow = 2
        };
    } // namespace assertAction

    DOCTEST_INTERFACE void failed_out_of_a_testing_context(const AssertData& ad);

    DOCTEST_INTERFACE bool decomp_assert(assertType::Enum at, const char* file, int line,
                                         const char* expr, const Result& result);

#define DOCTEST_ASSERT_OUT_OF_TESTS(decomp)                                                   \
    do {                                                                                       \
        if(!is_running_in_test) {                                                              \
            if(failed) {                                                                       \
                ResultBuilder rb(at, file, line, expr);                                        \
                rb.m_failed = failed;                                                          \
                rb.m_decomp = decomp;                                                          \
                failed_out_of_a_testing_context(rb);                                           \
                if(isDebuggerActive() && !getContextOptions()->no_breaks)                      \
                    DOCTEST_BREAK_INTO_DEBUGGER();                                             \
                if(checkIfShouldThrow(at))                                                     \
                    throwException();                                                          \
            }                                                                                  \
            return !failed;                                                                    \
        }                                                                                      \
    } while(false)

#define DOCTEST_ASSERT_IN_TESTS(decomp)                                                       \
    ResultBuilder rb(at, file, line, expr);                                                    \
    rb.m_failed = failed;                                                                      \
    if(rb.m_failed || getContextOptions()->success)                                            \
        rb.m_decomp = decomp;                                                                  \
    if(rb.log())                                                                               \
        DOCTEST_BREAK_INTO_DEBUGGER();                                                         \
    if(rb.m_failed && checkIfShouldThrow(at))                                                  \
    throwException()

    template <int comparison, typename L, typename R>
    DOCTEST_NOINLINE bool binary_assert(assertType::Enum at, const char* file, int line,
                                        const char* expr, const DOCTEST_REF_WRAP(L) lhs,
                                        const DOCTEST_REF_WRAP(R) rhs) {
        bool failed = !RelationalComparator<comparison, L, R>()(lhs, rhs);

        // ###################################################################################
        // IF THE DEBUGGER BREAKS HERE - GO 1 LEVEL UP IN THE CALLSTACK FOR THE FAILING ASSERT
        // THIS IS THE EFFECT OF HAVING 'DOCTEST_CONFIG_SUPER_FAST_ASSERTS' DEFINED
        // ###################################################################################
        DOCTEST_ASSERT_OUT_OF_TESTS(stringifyBinaryExpr(lhs, ", ", rhs));
        DOCTEST_ASSERT_IN_TESTS(stringifyBinaryExpr(lhs, ", ", rhs));
        return !failed;
    }

    template <typename L>
    DOCTEST_NOINLINE bool unary_assert(assertType::Enum at, const char* file, int line,
                                       const char* expr, const DOCTEST_REF_WRAP(L) val) {
        bool failed = !val;

        if(at & assertType::is_false)
            failed = !failed;

        // ###################################################################################
        // IF THE DEBUGGER BREAKS HERE - GO 1 LEVEL UP IN THE CALLSTACK FOR THE FAILING ASSERT
        // THIS IS THE EFFECT OF HAVING 'DOCTEST_CONFIG_SUPER_FAST_ASSERTS' DEFINED
        // ###################################################################################
        DOCTEST_ASSERT_OUT_OF_TESTS((DOCTEST_STRINGIFY(val)));
        DOCTEST_ASSERT_IN_TESTS((DOCTEST_STRINGIFY(val)));
        return !failed;
    }

    struct DOCTEST_INTERFACE IExceptionTranslator
    {
        DOCTEST_DECLARE_INTERFACE(IExceptionTranslator)
        virtual bool translate(String&) const = 0;
    };

    template <typename T>
    class ExceptionTranslator : public IExceptionTranslator
    {
    public:
        explicit ExceptionTranslator(String (*translateFunction)(T))
                : m_translateFunction(translateFunction) {}

        bool translate(String& res) const override {
#ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
            try {
                throw; // lgtm [cpp/rethrow-no-exception]
                // cppcheck-suppress catchExceptionByValue
            } catch(const T& ex) {
                res = m_translateFunction(ex);
                return true;
            } catch(...) {}
#endif                              // DOCTEST_CONFIG_NO_EXCEPTIONS
            static_cast<void>(res); // to silence -Wunused-parameter
            return false;
        }

    private:
        String (*m_translateFunction)(T);
    };

    DOCTEST_INTERFACE void registerExceptionTranslatorImpl(const IExceptionTranslator* et);

    // ContextScope base class used to allow implementing methods of ContextScope
    // that don't depend on the template parameter in doctest.cpp.
    struct DOCTEST_INTERFACE ContextScopeBase : public IContextScope {
        ContextScopeBase(const ContextScopeBase&) = delete;

        ContextScopeBase& operator=(const ContextScopeBase&) = delete;
        ContextScopeBase& operator=(ContextScopeBase&&) = delete;

        ~ContextScopeBase() override = default;

    protected:
        ContextScopeBase();
        ContextScopeBase(ContextScopeBase&& other) noexcept;

        void destroy();
        bool need_to_destroy{true};
    };

    template <typename L> class ContextScope : public ContextScopeBase
    {
        L lambda_;

    public:
        explicit ContextScope(const L &lambda) : lambda_(lambda) {}
        explicit ContextScope(L&& lambda) : lambda_(static_cast<L&&>(lambda)) { }

        ContextScope(const ContextScope&) = delete;
        ContextScope(ContextScope&&) noexcept = default;

        ContextScope& operator=(const ContextScope&) = delete;
        ContextScope& operator=(ContextScope&&) = delete;

        void stringify(std::ostream* s) const override { lambda_(s); }

        ~ContextScope() override {
            if (need_to_destroy) {
                destroy();
            }
        }
    };

    struct DOCTEST_INTERFACE MessageBuilder : public MessageData
    {
        std::ostream* m_stream;
        bool          logged = false;

        MessageBuilder(const char* file, int line, assertType::Enum severity);

        MessageBuilder(const MessageBuilder&) = delete;
        MessageBuilder(MessageBuilder&&) = delete;

        MessageBuilder& operator=(const MessageBuilder&) = delete;
        MessageBuilder& operator=(MessageBuilder&&) = delete;

        ~MessageBuilder();

        // the preferred way of chaining parameters for stringification
DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4866)
        template <typename T>
        MessageBuilder& operator,(const T& in) {
            *m_stream « (DOCTEST_STRINGIFY(in));
            return *this;
        }
DOCTEST_MSVC_SUPPRESS_WARNING_POP

        // kept here just for backwards-compatibility - the comma operator should be preferred now
        template <typename T>
        MessageBuilder& operator«(const T& in) { return this->operator,(in); }

        // the `,` operator has the lowest operator precedence - if `«` is used by the user then
        // the `,` operator will be called last which is not what we want and thus the `*` operator
        // is used first (has higher operator precedence compared to `«`) so that we guarantee that
        // an operator of the MessageBuilder class is called first before the rest of the parameters
        template <typename T>
        MessageBuilder& operator*(const T& in) { return this->operator,(in); }

        bool log();
        void react();
    };

    template <typename L>
    ContextScope<L> MakeContextScope(const L &lambda) {
        return ContextScope<L>(lambda);
    }
} // namespace detail

#define DOCTEST_DEFINE_DECORATOR(name, type, def)                                                  \
    struct name                                                                                    \
    {                                                                                              \
        type data;                                                                                 \
        name(type in = def)                                                                        \
                : data(in) {}                                                                      \
        void fill(detail::TestCase& state) const { state.DOCTEST_CAT(m_, name) = data; }           \
        void fill(detail::TestSuite& state) const { state.DOCTEST_CAT(m_, name) = data; }          \
    }

DOCTEST_DEFINE_DECORATOR(test_suite, const char*, "");
DOCTEST_DEFINE_DECORATOR(description, const char*, "");
DOCTEST_DEFINE_DECORATOR(skip, bool, true);
DOCTEST_DEFINE_DECORATOR(no_breaks, bool, true);
DOCTEST_DEFINE_DECORATOR(no_output, bool, true);
DOCTEST_DEFINE_DECORATOR(timeout, double, 0);
DOCTEST_DEFINE_DECORATOR(may_fail, bool, true);
DOCTEST_DEFINE_DECORATOR(should_fail, bool, true);
DOCTEST_DEFINE_DECORATOR(expected_failures, int, 0);

template <typename T>
int registerExceptionTranslator(String (*translateFunction)(T)) {
    DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wexit-time-destructors")
    static detail::ExceptionTranslator<T> exceptionTranslator(translateFunction);
    DOCTEST_CLANG_SUPPRESS_WARNING_POP
    detail::registerExceptionTranslatorImpl(&exceptionTranslator);
    return 0;
}

} // namespace doctest

// in a separate namespace outside of doctest because the DOCTEST_TEST_SUITE macro
// introduces an anonymous namespace in which getCurrentTestSuite gets overridden
namespace doctest_detail_test_suite_ns {
DOCTEST_INTERFACE doctest::detail::TestSuite& getCurrentTestSuite();
} // namespace doctest_detail_test_suite_ns

namespace doctest {
#else  // DOCTEST_CONFIG_DISABLE
template <typename T>
int registerExceptionTranslator(String (*)(T)) {
    return 0;
}
#endif // DOCTEST_CONFIG_DISABLE

namespace detail {
    using assert_handler = void (*)(const AssertData&);
    struct ContextState;
} // namespace detail

class DOCTEST_INTERFACE Context
{
    detail::ContextState* p;

    void parseArgs(int argc, const char* const* argv, bool withDefaults = false);

public:
    explicit Context(int argc = 0, const char* const* argv = nullptr);

    Context(const Context&) = delete;
    Context(Context&&) = delete;

    Context& operator=(const Context&) = delete;
    Context& operator=(Context&&) = delete;

    ~Context(); // NOLINT(performance-trivially-destructible)

    void applyCommandLine(int argc, const char* const* argv);

    void addFilter(const char* filter, const char* value);
    void clearFilters();
    void setOption(const char* option, bool value);
    void setOption(const char* option, int value);
    void setOption(const char* option, const char* value);

    bool shouldExit();

    void setAsDefaultForAssertsOutOfTestCases();

    void setAssertHandler(detail::assert_handler ah);

    void setCout(std::ostream* out);

    int run();
};

namespace TestCaseFailureReason {
    enum Enum
    {
        None                     = 0,
        AssertFailure            = 1,   // an assertion has failed in the test case
        Exception                = 2,   // test case threw an exception
        Crash                    = 4,   // a crash...
        TooManyFailedAsserts     = 8,   // the abort-after option
        Timeout                  = 16,  // see the timeout decorator
        ShouldHaveFailedButDidnt = 32,  // see the should_fail decorator
        ShouldHaveFailedAndDid   = 64,  // see the should_fail decorator
        DidntFailExactlyNumTimes = 128, // see the expected_failures decorator
        FailedExactlyNumTimes    = 256, // see the expected_failures decorator
        CouldHaveFailedAndDid    = 512  // see the may_fail decorator
    };
} // namespace TestCaseFailureReason

struct DOCTEST_INTERFACE CurrentTestCaseStats
{
    int    numAssertsCurrentTest;
    int    numAssertsFailedCurrentTest;
    double seconds;
    int    failure_flags; // use TestCaseFailureReason::Enum
    bool   testCaseSuccess;
};

struct DOCTEST_INTERFACE TestCaseException
{
    String error_string;
    bool   is_crash;
};

struct DOCTEST_INTERFACE TestRunStats
{
    unsigned numTestCases;
    unsigned numTestCasesPassingFilters;
    unsigned numTestSuitesPassingFilters;
    unsigned numTestCasesFailed;
    int      numAsserts;
    int      numAssertsFailed;
};

struct QueryData
{
    const TestRunStats*  run_stats = nullptr;
    const TestCaseData** data      = nullptr;
    unsigned             num_data  = 0;
};

struct DOCTEST_INTERFACE IReporter
{
    // The constructor has to accept "const ContextOptions&" as a single argument
    // which has most of the options for the run + a pointer to the stdout stream
    // Reporter(const ContextOptions& in)

    // called when a query should be reported (listing test cases, printing the version, etc.)
    virtual void report_query(const QueryData&) = 0;

    // called when the whole test run starts
    virtual void test_run_start() = 0;
    // called when the whole test run ends (caching a pointer to the input doesn't make sense here)
    virtual void test_run_end(const TestRunStats&) = 0;

    // called when a test case is started (safe to cache a pointer to the input)
    virtual void test_case_start(const TestCaseData&) = 0;
    // called when a test case is reentered because of unfinished subcases (safe to cache a pointer to
the input)
    virtual void test_case_reenter(const TestCaseData&) = 0;
    // called when a test case has ended
    virtual void test_case_end(const CurrentTestCaseStats&) = 0;

    // called when an exception is thrown from the test case (or it crashes)
    virtual void test_case_exception(const TestCaseException&) = 0;

    // called whenever a subcase is entered (don't cache pointers to the input)
    virtual void subcase_start(const SubcaseSignature&) = 0;
    // called whenever a subcase is exited (don't cache pointers to the input)
    virtual void subcase_end() = 0;

    // called for each assert (don't cache pointers to the input)
    virtual void log_assert(const AssertData&) = 0;
    // called for each message (don't cache pointers to the input)
    virtual void log_message(const MessageData&) = 0;

    // called when a test case is skipped either because it doesn't pass the filters, has a skip
decorator
    // or isn't in the execution range (between first and last) (safe to cache a pointer to the input)
    virtual void test_case_skipped(const TestCaseData&) = 0;

    DOCTEST_DECLARE_INTERFACE(IReporter)

    // can obtain all currently active contexts and stringify them if one wishes to do so
    static int                         get_num_active_contexts();
    static const IContextScope* const* get_active_contexts();

    // can iterate through contexts which have been stringified automatically in their destructors
when an exception has been thrown
    static int            get_num_stringified_contexts();
    static const String* get_stringified_contexts();
};

namespace detail {
    using reporterCreatorFunc =  IReporter* (*)(const ContextOptions&);

    DOCTEST_INTERFACE void registerReporterImpl(const char* name, int prio, reporterCreatorFunc c,
bool isReporter);

    template <typename Reporter>
    IReporter* reporterCreator(const ContextOptions& o) {
        return new Reporter(o);
    }
} // namespace detail

template <typename Reporter>
int registerReporter(const char* name, int priority, bool isReporter) {
    detail::registerReporterImpl(name, priority, detail::reporterCreator<Reporter>, isReporter);
    return 0;
}
} // namespace doctest

#ifdef DOCTEST_CONFIG_ASSERTS_RETURN_VALUES
#define DOCTEST_FUNC_EMPTY [] { return false; }()
#else
#define DOCTEST_FUNC_EMPTY (void)0
#endif

// if registering is not disabled
#ifndef DOCTEST_CONFIG_DISABLE

#ifdef DOCTEST_CONFIG_ASSERTS_RETURN_VALUES
#define DOCTEST_FUNC_SCOPE_BEGIN [&]
#define DOCTEST_FUNC_SCOPE_END ()
#define DOCTEST_FUNC_SCOPE_RET(v) return v
#else
#define DOCTEST_FUNC_SCOPE_BEGIN do
#define DOCTEST_FUNC_SCOPE_END while(false)
#define DOCTEST_FUNC_SCOPE_RET(v) (void)0
#endif

// common code in asserts - for convenience
#define DOCTEST_ASSERT_LOG_REACT_RETURN(b)                                                         \
    if(b.log()) DOCTEST_BREAK_INTO_DEBUGGER();                                                     \
    b.react();                                                                                     \
    DOCTEST_FUNC_SCOPE_RET(!b.m_failed)

#ifdef DOCTEST_CONFIG_NO_TRY_CATCH_IN_ASSERTS
#define DOCTEST_WRAP_IN_TRY(x) x;
#else // DOCTEST_CONFIG_NO_TRY_CATCH_IN_ASSERTS
#define DOCTEST_WRAP_IN_TRY(x)                                                                     \
    try {                                                                                          \
        x;                                                                                         \
    } catch(...) { DOCTEST_RB.translateException(); }
#endif // DOCTEST_CONFIG_NO_TRY_CATCH_IN_ASSERTS

#ifdef DOCTEST_CONFIG_VOID_CAST_EXPRESSIONS
#define DOCTEST_CAST_TO_VOID(...)                                                                  \
    DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH("-Wuseless-cast")                                    \
    static_cast<void>(__VA_ARGS__);                                                             \
    DOCTEST_GCC_SUPPRESS_WARNING_POP
#else // DOCTEST_CONFIG_VOID_CAST_EXPRESSIONS
#define DOCTEST_CAST_TO_VOID(...) __VA_ARGS__;
#endif // DOCTEST_CONFIG_VOID_CAST_EXPRESSIONS

// registers the test by initializing a dummy var with a function
#define DOCTEST_REGISTER_FUNCTION(global_prefix, f, decorators)                                 \
    global_prefix DOCTEST_GLOBAL_NO_WARNINGS(DOCTEST_ANONYMOUS(DOCTEST_ANON_VAR_), /* NOLINT */    \
            doctest::detail::regTest(                                                           \
                    doctest::detail::TestCase(                                                  \
                            f, __FILE__, __LINE__,                                              \
                            doctest_detail_test_suite_ns::getCurrentTestSuite()) *              \
                    decorators))

#define DOCTEST_IMPLEMENT_FIXTURE(der, base, func, decorators)                                  \
    namespace { /* NOLINT */                                                                    \
        struct der : public base                                                                \
        {                                                                                       \
            void f();                                                                           \
        };                                                                                      \
        static DOCTEST_INLINE_NOINLINE void func() {                                            \
            der v;                                                                              \
            v.f();                                                                              \
        }                                                                                       \
        DOCTEST_REGISTER_FUNCTION(DOCTEST_EMPTY, func, decorators)                              \
    }                                                                                           \
    DOCTEST_INLINE_NOINLINE void der::f() // NOLINT(misc-definitions-in-headers)

#define DOCTEST_CREATE_AND_REGISTER_FUNCTION(f, decorators)                                     \
    static void f();                                                                            \
    DOCTEST_REGISTER_FUNCTION(DOCTEST_EMPTY, f, decorators)                                     \
    static void f()

#define DOCTEST_CREATE_AND_REGISTER_FUNCTION_IN_CLASS(f, proxy, decorators)                     \
    static doctest::detail::funcType proxy() { return f; }                                      \
    DOCTEST_REGISTER_FUNCTION(inline, proxy(), decorators)                                      \
    static void f()

// for registering tests
#define DOCTEST_TEST_CASE(decorators)                                                           \
    DOCTEST_CREATE_AND_REGISTER_FUNCTION(DOCTEST_ANONYMOUS(DOCTEST_ANON_FUNC_), decorators)

// for registering tests in classes - requires C++17 for inline variables!
#if DOCTEST_CPLUSPLUS >= 201703L
#define DOCTEST_TEST_CASE_CLASS(decorators)                                                     \
    DOCTEST_CREATE_AND_REGISTER_FUNCTION_IN_CLASS(DOCTEST_ANONYMOUS(DOCTEST_ANON_FUNC_),        \
                                                  DOCTEST_ANONYMOUS(DOCTEST_ANON_PROXY_),       \
                                                  decorators)
#else // DOCTEST_TEST_CASE_CLASS
#define DOCTEST_TEST_CASE_CLASS(...)                                                            \
    TEST_CASES_CAN_BE_REGISTERED_IN_CLASSES_ONLY_IN_CPP17_MODE_OR_WITH_VS_2017_OR_NEWER
#endif // DOCTEST_TEST_CASE_CLASS

// for registering tests with a fixture
#define DOCTEST_TEST_CASE_FIXTURE(c, decorators)                                                \
    DOCTEST_IMPLEMENT_FIXTURE(DOCTEST_ANONYMOUS(DOCTEST_ANON_CLASS_), c,                        \
                              DOCTEST_ANONYMOUS(DOCTEST_ANON_FUNC_), decorators)

// for converting types to strings without the <typeinfo> header and demangling
#define DOCTEST_TYPE_TO_STRING_AS(str, ...)                                                     \
    namespace doctest {                                                                         \
        template <>                                                                             \
        inline String toString<__VA_ARGS__>() {                                                 \
            return str;                                                                         \
        }                                                                                       \
    }                                                                                           \
    static_assert(true, "")

#define DOCTEST_TYPE_TO_STRING(...) DOCTEST_TYPE_TO_STRING_AS(#__VA_ARGS__, __VA_ARGS__)

#define DOCTEST_TEST_CASE_TEMPLATE_DEFINE_IMPL(dec, T, iter, func)                              \
    template <typename T>                                                                       \
    static void func();                                                                         \
    namespace { /* NOLINT */                                                                    \
        template <typename Tuple>                                                               \
        struct iter;                                                                            \
        template <typename Type, typename... Rest>                                              \
        struct iter<std::tuple<Type, Rest...»                                                  \
        {                                                                                       \
            iter(const char* file, unsigned line, int index) {                                  \
                doctest::detail::regTest(doctest::detail::TestCase(func<Type>, file, line,      \
                                            doctest_detail_test_suite_ns::getCurrentTestSuite(),   \
                                            doctest::toString<Type>(),                          \
                                            int(line) * 1000 + index)                           \
                                         * dec);                                                \
                iter<std::tuple<Rest...»(file, line, index + 1);                        \
            }                                                                           \
        };                                                                              \
        template <>                                                                     \
        struct iter<std::tuple<»                                                        \
        {                                                                               \
            iter(const char*, unsigned, int) {}                                         \
        };                                                                              \
    }                                                                                   \
    template <typename T>                                                               \
    static void func()

#define DOCTEST_TEST_CASE_TEMPLATE_DEFINE(dec, T, id)                                   \
    DOCTEST_TEST_CASE_TEMPLATE_DEFINE_IMPL(dec, T, DOCTEST_CAT(id, ITERATOR),           \
                                           DOCTEST_ANONYMOUS(DOCTEST_ANON_TMP_))

#define DOCTEST_TEST_CASE_TEMPLATE_INSTANTIATE_IMPL(id, anon, ...)                      \
    DOCTEST_GLOBAL_NO_WARNINGS(DOCTEST_CAT(anon, DUMMY), /* NOLINT(cert-err58-cpp,
fuchsia-statically-constructed-objects) */ \
        doctest::detail::instantiationHelper(                                           \
            DOCTEST_CAT(id, ITERATOR)<__VA_ARGS__>(__FILE__, __LINE__, 0)))

#define DOCTEST_TEST_CASE_TEMPLATE_INVOKE(id, ...)                                      \
    DOCTEST_TEST_CASE_TEMPLATE_INSTANTIATE_IMPL(id, DOCTEST_ANONYMOUS(DOCTEST_ANON_TMP_),
std::tuple<__VA_ARGS__>) \
    static_assert(true, "")

#define DOCTEST_TEST_CASE_TEMPLATE_APPLY(id, ...)                                       \
    DOCTEST_TEST_CASE_TEMPLATE_INSTANTIATE_IMPL(id, DOCTEST_ANONYMOUS(DOCTEST_ANON_TMP_), __VA_ARGS__)
\
    static_assert(true, "")

#define DOCTEST_TEST_CASE_TEMPLATE_IMPL(dec, T, anon, ...)                              \
    DOCTEST_TEST_CASE_TEMPLATE_DEFINE_IMPL(dec, T, DOCTEST_CAT(anon, ITERATOR), anon);  \
    DOCTEST_TEST_CASE_TEMPLATE_INSTANTIATE_IMPL(anon, anon, std::tuple<__VA_ARGS__>)    \
    template <typename T>                                                               \
    static void anon()

#define DOCTEST_TEST_CASE_TEMPLATE(dec, T, ...)                                         \
    DOCTEST_TEST_CASE_TEMPLATE_IMPL(dec, T, DOCTEST_ANONYMOUS(DOCTEST_ANON_TMP_), __VA_ARGS__)

// for subcases
#define DOCTEST_SUBCASE(name)                                                           \
    if(const doctest::detail::Subcase & DOCTEST_ANONYMOUS(DOCTEST_ANON_SUBCASE_) DOCTEST_UNUSED =  \
               doctest::detail::Subcase(name, __FILE__, __LINE__))

// for grouping tests in test suites by using code blocks
#define DOCTEST_TEST_SUITE_IMPL(decorators, ns_name)                                    \
    namespace ns_name { namespace doctest_detail_test_suite_ns {                        \
            static DOCTEST_NOINLINE doctest::detail::TestSuite& getCurrentTestSuite() noexcept {   \
                DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4640)                           \
                DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wexit-time-destructors")     \
                DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH("-Wmissing-field-initializers")  \
                static doctest::detail::TestSuite data{};                               \
                static bool                        inited = false;                      \
                DOCTEST_MSVC_SUPPRESS_WARNING_POP                                       \
                DOCTEST_CLANG_SUPPRESS_WARNING_POP                                      \
                DOCTEST_GCC_SUPPRESS_WARNING_POP                                        \
                if(!inited) {                                                           \
                    data* decorators;                                                   \
                    inited = true;                                                      \
                }                                                                       \
                return data;                                                            \
            }                                                                           \
        }                                                                               \
    }                                                                                   \
    namespace ns_name

#define DOCTEST_TEST_SUITE(decorators)                                                  \
    DOCTEST_TEST_SUITE_IMPL(decorators, DOCTEST_ANONYMOUS(DOCTEST_ANON_SUITE_))

// for starting a testsuite block
#define DOCTEST_TEST_SUITE_BEGIN(decorators)                                            \
    DOCTEST_GLOBAL_NO_WARNINGS(DOCTEST_ANONYMOUS(DOCTEST_ANON_VAR_), /* NOLINT(cert-err58-cpp) */  \
            doctest::detail::setTestSuite(doctest::detail::TestSuite() * decorators))  \
    static_assert(true, "")

// for ending a testsuite block
#define DOCTEST_TEST_SUITE_END                                                          \
    DOCTEST_GLOBAL_NO_WARNINGS(DOCTEST_ANONYMOUS(DOCTEST_ANON_VAR_), /* NOLINT(cert-err58-cpp) */  \
            doctest::detail::setTestSuite(doctest::detail::TestSuite() * ""))           \
    using DOCTEST_ANONYMOUS(DOCTEST_ANON_FOR_SEMICOLON_) = int

// for registering exception translators
#define DOCTEST_REGISTER_EXCEPTION_TRANSLATOR_IMPL(translatorName, signature)           \
    inline doctest::String translatorName(signature);                                   \
    DOCTEST_GLOBAL_NO_WARNINGS(DOCTEST_ANONYMOUS(DOCTEST_ANON_TRANSLATOR_), /* NOLINT(cert-err58-cpp)
*/ \
            doctest::registerExceptionTranslator(translatorName))                                   \
    doctest::String translatorName(signature)

#define DOCTEST_REGISTER_EXCEPTION_TRANSLATOR(signature)                                            \
    DOCTEST_REGISTER_EXCEPTION_TRANSLATOR_IMPL(DOCTEST_ANONYMOUS(DOCTEST_ANON_TRANSLATOR_),         \
                                               signature)

// for registering reporters
#define DOCTEST_REGISTER_REPORTER(name, priority, reporter)                                         \
    DOCTEST_GLOBAL_NO_WARNINGS(DOCTEST_ANONYMOUS(DOCTEST_ANON_REPORTER_), /* NOLINT(cert-err58-cpp) */
\
            doctest::registerReporter<reporter>(name, priority, true))                              \
    static_assert(true, "")

// for registering listeners
#define DOCTEST_REGISTER_LISTENER(name, priority, reporter)                                         \
    DOCTEST_GLOBAL_NO_WARNINGS(DOCTEST_ANONYMOUS(DOCTEST_ANON_REPORTER_), /* NOLINT(cert-err58-cpp) */
\
            doctest::registerReporter<reporter>(name, priority, false))                             \
    static_assert(true, "")

// clang-format off
// for logging - disabling formatting because it's important to have these on 2 separate lines - see
PR #557
#define DOCTEST_INFO(...)                                                                           \
    DOCTEST_INFO_IMPL(DOCTEST_ANONYMOUS(DOCTEST_CAPTURE_),                                          \
                      DOCTEST_ANONYMOUS(DOCTEST_CAPTURE_OTHER_),                                    \
                      __VA_ARGS__)
// clang-format on

#define DOCTEST_INFO_IMPL(mb_name, s_name, ...)                                       \
    auto DOCTEST_ANONYMOUS(DOCTEST_CAPTURE_) = doctest::detail::MakeContextScope(                   \
        [&](std::ostream* s_name) {                                                                 \
        doctest::detail::MessageBuilder mb_name(__FILE__, __LINE__, doctest::assertType::is_warn); \
        mb_name.m_stream = s_name;                                                                  \
        mb_name * __VA_ARGS__;                                                                      \
    })

#define DOCTEST_CAPTURE(x) DOCTEST_INFO(#x " := ", x)

#define DOCTEST_ADD_AT_IMPL(type, file, line, mb, ...)                                              \
    DOCTEST_FUNC_SCOPE_BEGIN {                                                                      \
        doctest::detail::MessageBuilder mb(file, line, doctest::assertType::type);                  \
        mb * __VA_ARGS__;                                                                           \
        if(mb.log())                                                                                \
            DOCTEST_BREAK_INTO_DEBUGGER();                                                          \
        mb.react();                                                                                 \
    } DOCTEST_FUNC_SCOPE_END

// clang-format off
#define DOCTEST_ADD_MESSAGE_AT(file, line, ...) DOCTEST_ADD_AT_IMPL(is_warn, file, line,
DOCTEST_ANONYMOUS(DOCTEST_MESSAGE_), __VA_ARGS__)
#define DOCTEST_ADD_FAIL_CHECK_AT(file, line, ...) DOCTEST_ADD_AT_IMPL(is_check, file, line,
DOCTEST_ANONYMOUS(DOCTEST_MESSAGE_), __VA_ARGS__)
#define DOCTEST_ADD_FAIL_AT(file, line, ...) DOCTEST_ADD_AT_IMPL(is_require, file, line,
DOCTEST_ANONYMOUS(DOCTEST_MESSAGE_), __VA_ARGS__)
// clang-format on

#define DOCTEST_MESSAGE(...) DOCTEST_ADD_MESSAGE_AT(__FILE__, __LINE__, __VA_ARGS__)
#define DOCTEST_FAIL_CHECK(...) DOCTEST_ADD_FAIL_CHECK_AT(__FILE__, __LINE__, __VA_ARGS__)
#define DOCTEST_FAIL(...) DOCTEST_ADD_FAIL_AT(__FILE__, __LINE__, __VA_ARGS__)

#define DOCTEST_TO_LVALUE(...) __VA_ARGS__ // Not removed to keep backwards compatibility.

#ifndef DOCTEST_CONFIG_SUPER_FAST_ASSERTS

#define DOCTEST_ASSERT_IMPLEMENT_2(assert_type, ...)                                                \
    DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Woverloaded-shift-op-parentheses")                   \
    /* NOLINTNEXTLINE(clang-analyzer-cplusplus.NewDeleteLeaks) */                                   \
    doctest::detail::ResultBuilder DOCTEST_RB(doctest::assertType::assert_type, __FILE__,           \
                                               __LINE__, #__VA_ARGS__);                             \
    DOCTEST_WRAP_IN_TRY(DOCTEST_RB.setResult(                                                       \
            doctest::detail::ExpressionDecomposer(doctest::assertType::assert_type)                 \
            « __VA_ARGS__)) /* NOLINTNEXTLINE(clang-analyzer-cplusplus.NewDeleteLeaks) */         \
    DOCTEST_ASSERT_LOG_REACT_RETURN(DOCTEST_RB)                                                     \
    DOCTEST_CLANG_SUPPRESS_WARNING_POP

#define DOCTEST_ASSERT_IMPLEMENT_1(assert_type, ...)                                                \
    DOCTEST_FUNC_SCOPE_BEGIN {                                                                      \
        DOCTEST_ASSERT_IMPLEMENT_2(assert_type, __VA_ARGS__);                                       \
    } DOCTEST_FUNC_SCOPE_END // NOLINT(clang-analyzer-cplusplus.NewDeleteLeaks)

#define DOCTEST_BINARY_ASSERT(assert_type, comp, ...)                                               \
    DOCTEST_FUNC_SCOPE_BEGIN {                                                                      \
        doctest::detail::ResultBuilder DOCTEST_RB(doctest::assertType::assert_type, __FILE__,       \
                                                   __LINE__, #__VA_ARGS__);                         \
        DOCTEST_WRAP_IN_TRY(                                                                        \
                DOCTEST_RB.binary_assert<doctest::detail::binaryAssertComparison::comp>(            \
                        __VA_ARGS__))                                                               \
        DOCTEST_ASSERT_LOG_REACT_RETURN(DOCTEST_RB);                                                \
    } DOCTEST_FUNC_SCOPE_END

#define DOCTEST_UNARY_ASSERT(assert_type, ...)                                                      \
    DOCTEST_FUNC_SCOPE_BEGIN {                                                                      \
        doctest::detail::ResultBuilder DOCTEST_RB(doctest::assertType::assert_type, __FILE__,       \
                                                   __LINE__, #__VA_ARGS__);                         \
        DOCTEST_WRAP_IN_TRY(DOCTEST_RB.unary_assert(__VA_ARGS__))                                   \
        DOCTEST_ASSERT_LOG_REACT_RETURN(DOCTEST_RB);                                                \
    } DOCTEST_FUNC_SCOPE_END

#else // DOCTEST_CONFIG_SUPER_FAST_ASSERTS

// necessary for <ASSERT>_MESSAGE
#define DOCTEST_ASSERT_IMPLEMENT_2 DOCTEST_ASSERT_IMPLEMENT_1

#define DOCTEST_ASSERT_IMPLEMENT_1(assert_type, ...)                                                \
    DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Woverloaded-shift-op-parentheses")                   \
    doctest::detail::decomp_assert(                                                                 \
            doctest::assertType::assert_type, __FILE__, __LINE__, #__VA_ARGS__,                     \
            doctest::detail::ExpressionDecomposer(doctest::assertType::assert_type)                 \
                    « __VA_ARGS__) DOCTEST_CLANG_SUPPRESS_WARNING_POP

#define DOCTEST_BINARY_ASSERT(assert_type, comparison, ...)                                         \
    doctest::detail::binary_assert<doctest::detail::binaryAssertComparison::comparison>(            \
            doctest::assertType::assert_type, __FILE__, __LINE__, #__VA_ARGS__, __VA_ARGS__)

#define DOCTEST_UNARY_ASSERT(assert_type, ...)                                                      \
    doctest::detail::unary_assert(doctest::assertType::assert_type, __FILE__, __LINE__,             \
                                  #__VA_ARGS__, __VA_ARGS__)

#endif // DOCTEST_CONFIG_SUPER_FAST_ASSERTS

#define DOCTEST_WARN(...) DOCTEST_ASSERT_IMPLEMENT_1(DT_WARN, __VA_ARGS__)
#define DOCTEST_CHECK(...) DOCTEST_ASSERT_IMPLEMENT_1(DT_CHECK, __VA_ARGS__)
#define DOCTEST_REQUIRE(...) DOCTEST_ASSERT_IMPLEMENT_1(DT_REQUIRE, __VA_ARGS__)
#define DOCTEST_WARN_FALSE(...) DOCTEST_ASSERT_IMPLEMENT_1(DT_WARN_FALSE, __VA_ARGS__)
#define DOCTEST_CHECK_FALSE(...) DOCTEST_ASSERT_IMPLEMENT_1(DT_CHECK_FALSE, __VA_ARGS__)
#define DOCTEST_REQUIRE_FALSE(...) DOCTEST_ASSERT_IMPLEMENT_1(DT_REQUIRE_FALSE, __VA_ARGS__)

// clang-format off
#define DOCTEST_WARN_MESSAGE(cond, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
DOCTEST_ASSERT_IMPLEMENT_2(DT_WARN, cond); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_CHECK_MESSAGE(cond, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
DOCTEST_ASSERT_IMPLEMENT_2(DT_CHECK, cond); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_REQUIRE_MESSAGE(cond, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
DOCTEST_ASSERT_IMPLEMENT_2(DT_REQUIRE, cond); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_WARN_FALSE_MESSAGE(cond, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
DOCTEST_ASSERT_IMPLEMENT_2(DT_WARN_FALSE, cond); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_CHECK_FALSE_MESSAGE(cond, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
DOCTEST_ASSERT_IMPLEMENT_2(DT_CHECK_FALSE, cond); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_REQUIRE_FALSE_MESSAGE(cond, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
DOCTEST_ASSERT_IMPLEMENT_2(DT_REQUIRE_FALSE, cond); } DOCTEST_FUNC_SCOPE_END
// clang-format on

#define DOCTEST_WARN_EQ(...) DOCTEST_BINARY_ASSERT(DT_WARN_EQ, eq, __VA_ARGS__)
#define DOCTEST_CHECK_EQ(...) DOCTEST_BINARY_ASSERT(DT_CHECK_EQ, eq, __VA_ARGS__)
#define DOCTEST_REQUIRE_EQ(...) DOCTEST_BINARY_ASSERT(DT_REQUIRE_EQ, eq, __VA_ARGS__)
#define DOCTEST_WARN_NE(...) DOCTEST_BINARY_ASSERT(DT_WARN_NE, ne, __VA_ARGS__)
#define DOCTEST_CHECK_NE(...) DOCTEST_BINARY_ASSERT(DT_CHECK_NE, ne, __VA_ARGS__)
#define DOCTEST_REQUIRE_NE(...) DOCTEST_BINARY_ASSERT(DT_REQUIRE_NE, ne, __VA_ARGS__)
#define DOCTEST_WARN_GT(...) DOCTEST_BINARY_ASSERT(DT_WARN_GT, gt, __VA_ARGS__)
#define DOCTEST_CHECK_GT(...) DOCTEST_BINARY_ASSERT(DT_CHECK_GT, gt, __VA_ARGS__)
#define DOCTEST_REQUIRE_GT(...) DOCTEST_BINARY_ASSERT(DT_REQUIRE_GT, gt, __VA_ARGS__)
#define DOCTEST_WARN_LT(...) DOCTEST_BINARY_ASSERT(DT_WARN_LT, lt, __VA_ARGS__)
#define DOCTEST_CHECK_LT(...) DOCTEST_BINARY_ASSERT(DT_CHECK_LT, lt, __VA_ARGS__)
#define DOCTEST_REQUIRE_LT(...) DOCTEST_BINARY_ASSERT(DT_REQUIRE_LT, lt, __VA_ARGS__)
#define DOCTEST_WARN_GE(...) DOCTEST_BINARY_ASSERT(DT_WARN_GE, ge, __VA_ARGS__)
#define DOCTEST_CHECK_GE(...) DOCTEST_BINARY_ASSERT(DT_CHECK_GE, ge, __VA_ARGS__)
#define DOCTEST_REQUIRE_GE(...) DOCTEST_BINARY_ASSERT(DT_REQUIRE_GE, ge, __VA_ARGS__)
#define DOCTEST_WARN_LE(...) DOCTEST_BINARY_ASSERT(DT_WARN_LE, le, __VA_ARGS__)
#define DOCTEST_CHECK_LE(...) DOCTEST_BINARY_ASSERT(DT_CHECK_LE, le, __VA_ARGS__)
#define DOCTEST_REQUIRE_LE(...) DOCTEST_BINARY_ASSERT(DT_REQUIRE_LE, le, __VA_ARGS__)

#define DOCTEST_WARN_UNARY(...) DOCTEST_UNARY_ASSERT(DT_WARN_UNARY, __VA_ARGS__)
#define DOCTEST_CHECK_UNARY(...) DOCTEST_UNARY_ASSERT(DT_CHECK_UNARY, __VA_ARGS__)
#define DOCTEST_REQUIRE_UNARY(...) DOCTEST_UNARY_ASSERT(DT_REQUIRE_UNARY, __VA_ARGS__)
#define DOCTEST_WARN_UNARY_FALSE(...) DOCTEST_UNARY_ASSERT(DT_WARN_UNARY_FALSE, __VA_ARGS__)
#define DOCTEST_CHECK_UNARY_FALSE(...) DOCTEST_UNARY_ASSERT(DT_CHECK_UNARY_FALSE, __VA_ARGS__)
#define DOCTEST_REQUIRE_UNARY_FALSE(...) DOCTEST_UNARY_ASSERT(DT_REQUIRE_UNARY_FALSE, __VA_ARGS__)

#ifndef DOCTEST_CONFIG_NO_EXCEPTIONS

#define DOCTEST_ASSERT_THROWS_AS(expr, assert_type, message, ...)                                   \
    DOCTEST_FUNC_SCOPE_BEGIN {                                                                         \
        if(!doctest::getContextOptions()->no_throw) {                                                  \
            doctest::detail::ResultBuilder DOCTEST_RB(doctest::assertType::assert_type, __FILE__,  \
                                                       __LINE__, #expr, #__VA_ARGS__, message);   \
            try {                                                                                      \
                DOCTEST_CAST_TO_VOID(expr)                                                             \
            } catch(const typename doctest::detail::types::remove_const<                              \
                    typename doctest::detail::types::remove_reference<__VA_ARGS__>::type>::type&) {\
                DOCTEST_RB.translateException();                                                       \
                DOCTEST_RB.m_threw_as = true;                                                          \
            } catch(...) { DOCTEST_RB.translateException(); }                                          \
            DOCTEST_ASSERT_LOG_REACT_RETURN(DOCTEST_RB);                                              \
        } else { /* NOLINT(*-else-after-return) */                                                    \
            DOCTEST_FUNC_SCOPE_RET(false);                                                             \
        }                                                                                              \
    } DOCTEST_FUNC_SCOPE_END

#define DOCTEST_ASSERT_THROWS_WITH(expr, expr_str, assert_type, ...)                                   \
    DOCTEST_FUNC_SCOPE_BEGIN {                                                                         \
        if(!doctest::getContextOptions()->no_throw) {                                                  \
            doctest::detail::ResultBuilder DOCTEST_RB(doctest::assertType::assert_type, __FILE__,  \
                                                       __LINE__, expr_str, "", __VA_ARGS__);       \
            try {                                                                                      \
                DOCTEST_CAST_TO_VOID(expr)                                                             \
            } catch(...) { DOCTEST_RB.translateException(); }                                          \
            DOCTEST_ASSERT_LOG_REACT_RETURN(DOCTEST_RB);                                              \
        } else { /* NOLINT(*-else-after-return) */                                                    \
           DOCTEST_FUNC_SCOPE_RET(false);                                                              \
        }                                                                                              \
    } DOCTEST_FUNC_SCOPE_END

#define DOCTEST_ASSERT_NOTHROW(assert_type, ...)                                                       \
    DOCTEST_FUNC_SCOPE_BEGIN {                                                                         \
        doctest::detail::ResultBuilder DOCTEST_RB(doctest::assertType::assert_type, __FILE__,      \
                                                   __LINE__, #__VA_ARGS__);                           \
        try {                                                                                          \
            DOCTEST_CAST_TO_VOID(__VA_ARGS__)                                                          \
        } catch(...) { DOCTEST_RB.translateException(); }                                              \
        DOCTEST_ASSERT_LOG_REACT_RETURN(DOCTEST_RB);                                                  \
    } DOCTEST_FUNC_SCOPE_END

// clang-format off
#define DOCTEST_WARN_THROWS(...) DOCTEST_ASSERT_THROWS_WITH((__VA_ARGS__), #__VA_ARGS__,
DT_WARN_THROWS, "")
#define DOCTEST_CHECK_THROWS(...) DOCTEST_ASSERT_THROWS_WITH((__VA_ARGS__), #__VA_ARGS__,
DT_CHECK_THROWS, "")
#define DOCTEST_REQUIRE_THROWS(...) DOCTEST_ASSERT_THROWS_WITH((__VA_ARGS__), #__VA_ARGS__,
DT_REQUIRE_THROWS, "")

#define DOCTEST_WARN_THROWS_AS(expr, ...) DOCTEST_ASSERT_THROWS_AS(expr, DT_WARN_THROWS_AS, "",
__VA_ARGS__)
#define DOCTEST_CHECK_THROWS_AS(expr, ...) DOCTEST_ASSERT_THROWS_AS(expr, DT_CHECK_THROWS_AS, "",
__VA_ARGS__)
#define DOCTEST_REQUIRE_THROWS_AS(expr, ...) DOCTEST_ASSERT_THROWS_AS(expr, DT_REQUIRE_THROWS_AS, "",
__VA_ARGS__)

#define DOCTEST_WARN_THROWS_WITH(expr, ...) DOCTEST_ASSERT_THROWS_WITH(expr, #expr,
DT_WARN_THROWS_WITH, __VA_ARGS__)
#define DOCTEST_CHECK_THROWS_WITH(expr, ...) DOCTEST_ASSERT_THROWS_WITH(expr, #expr,
DT_CHECK_THROWS_WITH, __VA_ARGS__)
#define DOCTEST_REQUIRE_THROWS_WITH(expr, ...) DOCTEST_ASSERT_THROWS_WITH(expr, #expr,
DT_REQUIRE_THROWS_WITH, __VA_ARGS__)

#define DOCTEST_WARN_THROWS_WITH_AS(expr, message, ...) DOCTEST_ASSERT_THROWS_AS(expr,
DT_WARN_THROWS_WITH_AS, message, __VA_ARGS__)
#define DOCTEST_CHECK_THROWS_WITH_AS(expr, message, ...) DOCTEST_ASSERT_THROWS_AS(expr,
DT_CHECK_THROWS_WITH_AS, message, __VA_ARGS__)
#define DOCTEST_REQUIRE_THROWS_WITH_AS(expr, message, ...) DOCTEST_ASSERT_THROWS_AS(expr,
DT_REQUIRE_THROWS_WITH_AS, message, __VA_ARGS__)

#define DOCTEST_WARN_NOTHROW(...) DOCTEST_ASSERT_NOTHROW(DT_WARN_NOTHROW, __VA_ARGS__)
#define DOCTEST_CHECK_NOTHROW(...) DOCTEST_ASSERT_NOTHROW(DT_CHECK_NOTHROW, __VA_ARGS__)
#define DOCTEST_REQUIRE_NOTHROW(...) DOCTEST_ASSERT_NOTHROW(DT_REQUIRE_NOTHROW, __VA_ARGS__)

#define DOCTEST_WARN_THROWS_MESSAGE(expr, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
DOCTEST_WARN_THROWS(expr); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_CHECK_THROWS_MESSAGE(expr, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
DOCTEST_CHECK_THROWS(expr); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_REQUIRE_THROWS_MESSAGE(expr, ...) DOCTEST_FUNC_SCOPE_BEGIN {
DOCTEST_INFO(__VA_ARGS__); DOCTEST_REQUIRE_THROWS(expr); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_WARN_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_FUNC_SCOPE_BEGIN {
DOCTEST_INFO(__VA_ARGS__); DOCTEST_WARN_THROWS_AS(expr, ex); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_CHECK_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_FUNC_SCOPE_BEGIN {
DOCTEST_INFO(__VA_ARGS__); DOCTEST_CHECK_THROWS_AS(expr, ex); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_REQUIRE_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_FUNC_SCOPE_BEGIN {
DOCTEST_INFO(__VA_ARGS__); DOCTEST_REQUIRE_THROWS_AS(expr, ex); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_WARN_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_FUNC_SCOPE_BEGIN {
```

```
DOCTEST_INFO(__VA_ARGS__); DOCTEST_WARN_THROWS_WITH(expr, with); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_CHECK_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__); DOCTEST_CHECK_THROWS_WITH(expr, with); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_REQUIRE_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__); DOCTEST_REQUIRE_THROWS_WITH(expr, with); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_WARN_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__); DOCTEST_WARN_THROWS_WITH_AS(expr, with, ex); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_CHECK_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__); DOCTEST_CHECK_THROWS_WITH_AS(expr, with, ex); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_REQUIRE_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__); DOCTEST_REQUIRE_THROWS_WITH_AS(expr, with, ex); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_WARN_NOTHROW_MESSAGE(expr, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__); DOCTEST_WARN_NOTHROW(expr); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_CHECK_NOTHROW_MESSAGE(expr, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__); DOCTEST_CHECK_NOTHROW(expr); } DOCTEST_FUNC_SCOPE_END
#define DOCTEST_REQUIRE_NOTHROW_MESSAGE(expr, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__); DOCTEST_REQUIRE_NOTHROW(expr); } DOCTEST_FUNC_SCOPE_END
// clang-format on

#endif // DOCTEST_CONFIG_NO_EXCEPTIONS

// =================================================================================================
// == WHAT FOLLOWS IS VERSIONS OF THE MACROS THAT DO NOT DO ANY REGISTERING!                     ==
// == THIS CAN BE ENABLED BY DEFINING DOCTEST_CONFIG_DISABLE GLOBALLY!                           ==
// =================================================================================================
#else // DOCTEST_CONFIG_DISABLE

#define DOCTEST_IMPLEMENT_FIXTURE(der, base, func, name)                                           \
    namespace /* NOLINT */ {                                                                       \
        template <typename DOCTEST_UNUSED_TEMPLATE_TYPE>                                           \
        struct der : public base                                                                   \
        { void f(); };                                                                             \
    }                                                                                              \
    template <typename DOCTEST_UNUSED_TEMPLATE_TYPE>                                               \
    inline void der<DOCTEST_UNUSED_TEMPLATE_TYPE>::f()

#define DOCTEST_CREATE_AND_REGISTER_FUNCTION(f, name)                                              \
    template <typename DOCTEST_UNUSED_TEMPLATE_TYPE>                                               \
    static inline void f()

// for registering tests
#define DOCTEST_TEST_CASE(name)                                                                    \
    DOCTEST_CREATE_AND_REGISTER_FUNCTION(DOCTEST_ANONYMOUS(DOCTEST_ANON_FUNC_), name)

// for registering tests in classes
#define DOCTEST_TEST_CASE_CLASS(name)                                                              \
    DOCTEST_CREATE_AND_REGISTER_FUNCTION(DOCTEST_ANONYMOUS(DOCTEST_ANON_FUNC_), name)

// for registering tests with a fixture
#define DOCTEST_TEST_CASE_FIXTURE(x, name)                                                         \
    DOCTEST_IMPLEMENT_FIXTURE(DOCTEST_ANONYMOUS(DOCTEST_ANON_CLASS_), x,                           \
                              DOCTEST_ANONYMOUS(DOCTEST_ANON_FUNC_), name)

// for converting types to strings without the <typeinfo> header and demangling
#define DOCTEST_TYPE_TO_STRING_AS(str, ...) static_assert(true, "")
#define DOCTEST_TYPE_TO_STRING(...) static_assert(true, "")

// for typed tests
#define DOCTEST_TEST_CASE_TEMPLATE(name, type, ...)                                                \
    template <typename type>                                                                       \
    inline void DOCTEST_ANONYMOUS(DOCTEST_ANON_TMP_)()

#define DOCTEST_TEST_CASE_TEMPLATE_DEFINE(name, type, id)                                          \
    template <typename type>                                                                       \
    inline void DOCTEST_ANONYMOUS(DOCTEST_ANON_TMP_)()

#define DOCTEST_TEST_CASE_TEMPLATE_INVOKE(id, ...) static_assert(true, "")
#define DOCTEST_TEST_CASE_TEMPLATE_APPLY(id, ...) static_assert(true, "")

// for subcases
#define DOCTEST_SUBCASE(name)

// for a testsuite block
#define DOCTEST_TEST_SUITE(name) namespace // NOLINT

// for starting a testsuite block
#define DOCTEST_TEST_SUITE_BEGIN(name) static_assert(true, "")

// for ending a testsuite block
#define DOCTEST_TEST_SUITE_END using DOCTEST_ANONYMOUS(DOCTEST_ANON_FOR_SEMICOLON_) = int

#define DOCTEST_REGISTER_EXCEPTION_TRANSLATOR(signature)                                           \
    template <typename DOCTEST_UNUSED_TEMPLATE_TYPE>                                               \
    static inline doctest::String DOCTEST_ANONYMOUS(DOCTEST_ANON_TRANSLATOR_)(signature)

#define DOCTEST_REGISTER_REPORTER(name, priority, reporter)
#define DOCTEST_REGISTER_LISTENER(name, priority, reporter)
```

```

#define DOCTEST_INFO(...) (static_cast<void>(0))
#define DOCTEST_CAPTURE(x) (static_cast<void>(0))
#define DOCTEST_ADD_MESSAGE_AT(file, line, ...) (static_cast<void>(0))
#define DOCTEST_ADD_FAIL_CHECK_AT(file, line, ...) (static_cast<void>(0))
#define DOCTEST_ADD_FAIL_AT(file, line, ...) (static_cast<void>(0))
#define DOCTEST_MESSAGE(...) (static_cast<void>(0))
#define DOCTEST_FAIL_CHECK(...) (static_cast<void>(0))
#define DOCTEST_FAIL(...) (static_cast<void>(0))

#if defined(DOCTEST_CONFIG_EVALUATE_ASSERTS_EVEN_WHEN_DISABLED)                                    \
 && defined(DOCTEST_CONFIG_ASSERTS_RETURN_VALUES)

#define DOCTEST_WARN(...) [&] { return __VA_ARGS__; }()
#define DOCTEST_CHECK(...) [&] { return __VA_ARGS__; }()
#define DOCTEST_REQUIRE(...) [&] { return __VA_ARGS__; }()
#define DOCTEST_WARN_FALSE(...) [&] { return !(__VA_ARGS__); }()
#define DOCTEST_CHECK_FALSE(...) [&] { return !(__VA_ARGS__); }()
#define DOCTEST_REQUIRE_FALSE(...) [&] { return !(__VA_ARGS__); }()

#define DOCTEST_WARN_MESSAGE(cond, ...) [&] { return cond; }()
#define DOCTEST_CHECK_MESSAGE(cond, ...) [&] { return cond; }()
#define DOCTEST_REQUIRE_MESSAGE(cond, ...) [&] { return cond; }()
#define DOCTEST_WARN_FALSE_MESSAGE(cond, ...) [&] { return !(cond); }()
#define DOCTEST_CHECK_FALSE_MESSAGE(cond, ...) [&] { return !(cond); }()
#define DOCTEST_REQUIRE_FALSE_MESSAGE(cond, ...) [&] { return !(cond); }()

namespace doctest {
namespace detail {
#define DOCTEST_RELATIONAL_OP(name, op)                                                            \
    template <typename L, typename R>                                                              \
    bool name(const DOCTEST_REF_WRAP(L) lhs, const DOCTEST_REF_WRAP(R) rhs) { return lhs op rhs; }

    DOCTEST_RELATIONAL_OP(eq, ==)
    DOCTEST_RELATIONAL_OP(ne, !=)
    DOCTEST_RELATIONAL_OP(lt, <)
    DOCTEST_RELATIONAL_OP(gt, >)
    DOCTEST_RELATIONAL_OP(le, <=)
    DOCTEST_RELATIONAL_OP(ge, >=)
} // namespace detail
} // namespace doctest

#define DOCTEST_WARN_EQ(...) [&] { return doctest::detail::eq(__VA_ARGS__); }()
#define DOCTEST_CHECK_EQ(...) [&] { return doctest::detail::eq(__VA_ARGS__); }()
#define DOCTEST_REQUIRE_EQ(...) [&] { return doctest::detail::eq(__VA_ARGS__); }()
#define DOCTEST_WARN_NE(...) [&] { return doctest::detail::ne(__VA_ARGS__); }()
#define DOCTEST_CHECK_NE(...) [&] { return doctest::detail::ne(__VA_ARGS__); }()
#define DOCTEST_REQUIRE_NE(...) [&] { return doctest::detail::ne(__VA_ARGS__); }()
#define DOCTEST_WARN_LT(...) [&] { return doctest::detail::lt(__VA_ARGS__); }()
#define DOCTEST_CHECK_LT(...) [&] { return doctest::detail::lt(__VA_ARGS__); }()
#define DOCTEST_REQUIRE_LT(...) [&] { return doctest::detail::lt(__VA_ARGS__); }()
#define DOCTEST_WARN_GT(...) [&] { return doctest::detail::gt(__VA_ARGS__); }()
#define DOCTEST_CHECK_GT(...) [&] { return doctest::detail::gt(__VA_ARGS__); }()
#define DOCTEST_REQUIRE_GT(...) [&] { return doctest::detail::gt(__VA_ARGS__); }()
#define DOCTEST_WARN_LE(...) [&] { return doctest::detail::le(__VA_ARGS__); }()
#define DOCTEST_CHECK_LE(...) [&] { return doctest::detail::le(__VA_ARGS__); }()
#define DOCTEST_REQUIRE_LE(...) [&] { return doctest::detail::le(__VA_ARGS__); }()
#define DOCTEST_WARN_GE(...) [&] { return doctest::detail::ge(__VA_ARGS__); }()
#define DOCTEST_CHECK_GE(...) [&] { return doctest::detail::ge(__VA_ARGS__); }()
#define DOCTEST_REQUIRE_GE(...) [&] { return doctest::detail::ge(__VA_ARGS__); }()
#define DOCTEST_WARN_UNARY(...) [&] { return __VA_ARGS__; }()
#define DOCTEST_CHECK_UNARY(...) [&] { return __VA_ARGS__; }()
#define DOCTEST_REQUIRE_UNARY(...) [&] { return __VA_ARGS__; }()
#define DOCTEST_WARN_UNARY_FALSE(...) [&] { return !(__VA_ARGS__); }()
#define DOCTEST_CHECK_UNARY_FALSE(...) [&] { return !(__VA_ARGS__); }()
#define DOCTEST_REQUIRE_UNARY_FALSE(...) [&] { return !(__VA_ARGS__); }()

#ifndef DOCTEST_CONFIG_NO_EXCEPTIONS

#define DOCTEST_WARN_THROWS_WITH(expr, with, ...) [] { static_assert(false, "Exception translation is
not available when doctest is disabled."); return false; }()
#define DOCTEST_CHECK_THROWS_WITH(expr, with, ...) DOCTEST_WARN_THROWS_WITH(„)
#define DOCTEST_REQUIRE_THROWS_WITH(expr, with, ...) DOCTEST_WARN_THROWS_WITH(„)
#define DOCTEST_WARN_THROWS_WITH_AS(expr, with, ex, ...) DOCTEST_WARN_THROWS_WITH(„)
#define DOCTEST_CHECK_THROWS_WITH_AS(expr, with, ex, ...) DOCTEST_WARN_THROWS_WITH(„)
#define DOCTEST_REQUIRE_THROWS_WITH_AS(expr, with, ex, ...) DOCTEST_WARN_THROWS_WITH(„)

#define DOCTEST_WARN_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_WARN_THROWS_WITH(„)
#define DOCTEST_CHECK_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_WARN_THROWS_WITH(„)
#define DOCTEST_REQUIRE_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_WARN_THROWS_WITH(„)
#define DOCTEST_WARN_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_WARN_THROWS_WITH(„)
#define DOCTEST_CHECK_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_WARN_THROWS_WITH(„)
#define DOCTEST_REQUIRE_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_WARN_THROWS_WITH(„)

#define DOCTEST_WARN_THROWS(...) [&] { try { __VA_ARGS__; return false; } catch (...) { return true; }
}()
#define DOCTEST_CHECK_THROWS(...) [&] { try { __VA_ARGS__; return false; } catch (...) { return true;
} }()
#define DOCTEST_REQUIRE_THROWS(...) [&] { try { __VA_ARGS__; return false; } catch (...) { return
true; } }()
#define DOCTEST_WARN_THROWS_AS(expr, ...) [&] { try { expr; } catch (__VA_ARGS__) { return true; }
catch (...) { } return false; }()
#define DOCTEST_CHECK_THROWS_AS(expr, ...) [&] { try { expr; } catch (__VA_ARGS__) { return true; }
catch (...) { } return false; }()
#define DOCTEST_REQUIRE_THROWS_AS(expr, ...) [&] { try { expr; } catch (__VA_ARGS__) { return true; }
catch (...) { } return false; }()
#define DOCTEST_WARN_NOTHROW(...) [&] { try { __VA_ARGS__; return true; } catch (...) { return false;
} }()
#define DOCTEST_CHECK_NOTHROW(...) [&] { try { __VA_ARGS__; return true; } catch (...) { return false;
} }()
#define DOCTEST_REQUIRE_NOTHROW(...) [&] { try { __VA_ARGS__; return true; } catch (...) { return
false; } }()

#define DOCTEST_WARN_THROWS_MESSAGE(expr, ...) [&] { try { __VA_ARGS__; return false; } catch (...) {
return true; } }()
#define DOCTEST_CHECK_THROWS_MESSAGE(expr, ...) [&] { try { __VA_ARGS__; return false; } catch (...) {
return true; } }()
#define DOCTEST_REQUIRE_THROWS_MESSAGE(expr, ...) [&] { try { __VA_ARGS__; return false; } catch (...)
{ return true; } }()
#define DOCTEST_WARN_THROWS_AS_MESSAGE(expr, ex, ...) [&] { try { expr; } catch (__VA_ARGS__) { return
true; } catch (...) { } return false; }()
#define DOCTEST_CHECK_THROWS_AS_MESSAGE(expr, ex, ...) [&] { try { expr; } catch (__VA_ARGS__) {
return true; } catch (...) { } return false; }()
#define DOCTEST_REQUIRE_THROWS_AS_MESSAGE(expr, ex, ...) [&] { try { expr; } catch (__VA_ARGS__) {
return true; } catch (...) { } return false; }()
#define DOCTEST_WARN_NOTHROW_MESSAGE(expr, ...) [&] { try { __VA_ARGS__; return true; } catch (...) {
return false; } }()
#define DOCTEST_CHECK_NOTHROW_MESSAGE(expr, ...) [&] { try { __VA_ARGS__; return true; } catch (...) {
return false; } }()
#define DOCTEST_REQUIRE_NOTHROW_MESSAGE(expr, ...) [&] { try { __VA_ARGS__; return true; } catch (...)
{ return false; } }()

#endif // DOCTEST_CONFIG_NO_EXCEPTIONS

#else // DOCTEST_CONFIG_EVALUATE_ASSERTS_EVEN_WHEN_DISABLED

#define DOCTEST_WARN(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_WARN_FALSE(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_FALSE(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_FALSE(...) DOCTEST_FUNC_EMPTY

#define DOCTEST_WARN_MESSAGE(cond, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_MESSAGE(cond, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_MESSAGE(cond, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_WARN_FALSE_MESSAGE(cond, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_FALSE_MESSAGE(cond, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_FALSE_MESSAGE(cond, ...) DOCTEST_FUNC_EMPTY

#define DOCTEST_WARN_EQ(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_EQ(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_EQ(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_WARN_NE(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_NE(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_NE(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_WARN_GT(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_GT(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_GT(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_WARN_LT(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_LT(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_LT(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_WARN_GE(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_GE(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_GE(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_WARN_LE(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_LE(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_LE(...) DOCTEST_FUNC_EMPTY

#define DOCTEST_WARN_UNARY(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_UNARY(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_UNARY(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_WARN_UNARY_FALSE(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_UNARY_FALSE(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_UNARY_FALSE(...) DOCTEST_FUNC_EMPTY

#ifndef DOCTEST_CONFIG_NO_EXCEPTIONS

#define DOCTEST_WARN_THROWS(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_THROWS(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_THROWS(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_WARN_THROWS_AS(expr, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_THROWS_AS(expr, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_THROWS_AS(expr, ...) DOCTEST_FUNC_EMPTY
```

```
#define DOCTEST_WARN_THROWS_WITH(expr, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_THROWS_WITH(expr, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_THROWS_WITH(expr, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_WARN_THROWS_WITH_AS(expr, with, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_THROWS_WITH_AS(expr, with, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_THROWS_WITH_AS(expr, with, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_WARN_NOTHROW(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_NOTHROW(...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_NOTHROW(...) DOCTEST_FUNC_EMPTY

#define DOCTEST_WARN_THROWS_MESSAGE(expr, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_THROWS_MESSAGE(expr, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_THROWS_MESSAGE(expr, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_WARN_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_WARN_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_WARN_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_WARN_NOTHROW_MESSAGE(expr, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_CHECK_NOTHROW_MESSAGE(expr, ...) DOCTEST_FUNC_EMPTY
#define DOCTEST_REQUIRE_NOTHROW_MESSAGE(expr, ...) DOCTEST_FUNC_EMPTY

#endif // DOCTEST_CONFIG_NO_EXCEPTIONS

#endif // DOCTEST_CONFIG_EVALUATE_ASSERTS_EVEN_WHEN_DISABLED

#endif // DOCTEST_CONFIG_DISABLE

#ifdef DOCTEST_CONFIG_NO_EXCEPTIONS

#ifdef DOCTEST_CONFIG_NO_EXCEPTIONS_BUT_WITH_ALL_ASSERTS
#define DOCTEST_EXCEPTION_EMPTY_FUNC DOCTEST_FUNC_EMPTY
#else // DOCTEST_CONFIG_NO_EXCEPTIONS_BUT_WITH_ALL_ASSERTS
#define DOCTEST_EXCEPTION_EMPTY_FUNC [] { static_assert(false, "Exceptions are disabled! " \
    "Use DOCTEST_CONFIG_NO_EXCEPTIONS_BUT_WITH_ALL_ASSERTS if you want to compile with exceptions
disabled."); return false; }()

#undef DOCTEST_REQUIRE
#undef DOCTEST_REQUIRE_FALSE
#undef DOCTEST_REQUIRE_MESSAGE
#undef DOCTEST_REQUIRE_FALSE_MESSAGE
#undef DOCTEST_REQUIRE_EQ
#undef DOCTEST_REQUIRE_NE
#undef DOCTEST_REQUIRE_GT
#undef DOCTEST_REQUIRE_LT
#undef DOCTEST_REQUIRE_GE
#undef DOCTEST_REQUIRE_LE
#undef DOCTEST_REQUIRE_UNARY
#undef DOCTEST_REQUIRE_UNARY_FALSE

#define DOCTEST_REQUIRE DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_FALSE DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_MESSAGE DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_FALSE_MESSAGE DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_EQ DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_NE DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_GT DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_LT DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_GE DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_LE DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_UNARY DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_UNARY_FALSE DOCTEST_EXCEPTION_EMPTY_FUNC

#endif // DOCTEST_CONFIG_NO_EXCEPTIONS_BUT_WITH_ALL_ASSERTS

#define DOCTEST_WARN_THROWS(...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_CHECK_THROWS(...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_THROWS(...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_WARN_THROWS_AS(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_CHECK_THROWS_AS(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_THROWS_AS(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_WARN_THROWS_WITH(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_CHECK_THROWS_WITH(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_THROWS_WITH(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_WARN_THROWS_WITH_AS(expr, with, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_CHECK_THROWS_WITH_AS(expr, with, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_THROWS_WITH_AS(expr, with, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_WARN_NOTHROW(...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_CHECK_NOTHROW(...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_NOTHROW(...) DOCTEST_EXCEPTION_EMPTY_FUNC

#define DOCTEST_WARN_THROWS_MESSAGE(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_CHECK_THROWS_MESSAGE(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
```

```
#define DOCTEST_REQUIRE_THROWS_MESSAGE(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_WARN_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_CHECK_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_WARN_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_CHECK_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_WARN_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_CHECK_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_WARN_NOTHROW_MESSAGE(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_CHECK_NOTHROW_MESSAGE(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
#define DOCTEST_REQUIRE_NOTHROW_MESSAGE(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC

#endif // DOCTEST_CONFIG_NO_EXCEPTIONS

// clang-format off
// KEPT FOR BACKWARDS COMPATIBILITY - FORWARDING TO THE RIGHT MACROS
#define DOCTEST_FAST_WARN_EQ             DOCTEST_WARN_EQ
#define DOCTEST_FAST_CHECK_EQ            DOCTEST_CHECK_EQ
#define DOCTEST_FAST_REQUIRE_EQ          DOCTEST_REQUIRE_EQ
#define DOCTEST_FAST_WARN_NE             DOCTEST_WARN_NE
#define DOCTEST_FAST_CHECK_NE            DOCTEST_CHECK_NE
#define DOCTEST_FAST_REQUIRE_NE          DOCTEST_REQUIRE_NE
#define DOCTEST_FAST_WARN_GT             DOCTEST_WARN_GT
#define DOCTEST_FAST_CHECK_GT            DOCTEST_CHECK_GT
#define DOCTEST_FAST_REQUIRE_GT          DOCTEST_REQUIRE_GT
#define DOCTEST_FAST_WARN_LT             DOCTEST_WARN_LT
#define DOCTEST_FAST_CHECK_LT            DOCTEST_CHECK_LT
#define DOCTEST_FAST_REQUIRE_LT          DOCTEST_REQUIRE_LT
#define DOCTEST_FAST_WARN_GE             DOCTEST_WARN_GE
#define DOCTEST_FAST_CHECK_GE            DOCTEST_CHECK_GE
#define DOCTEST_FAST_REQUIRE_GE          DOCTEST_REQUIRE_GE
#define DOCTEST_FAST_WARN_LE             DOCTEST_WARN_LE
#define DOCTEST_FAST_CHECK_LE            DOCTEST_CHECK_LE
#define DOCTEST_FAST_REQUIRE_LE          DOCTEST_REQUIRE_LE

#define DOCTEST_FAST_WARN_UNARY          DOCTEST_WARN_UNARY
#define DOCTEST_FAST_CHECK_UNARY         DOCTEST_CHECK_UNARY
#define DOCTEST_FAST_REQUIRE_UNARY       DOCTEST_REQUIRE_UNARY
#define DOCTEST_FAST_WARN_UNARY_FALSE    DOCTEST_WARN_UNARY_FALSE
#define DOCTEST_FAST_CHECK_UNARY_FALSE   DOCTEST_CHECK_UNARY_FALSE
#define DOCTEST_FAST_REQUIRE_UNARY_FALSE DOCTEST_REQUIRE_UNARY_FALSE

#define DOCTEST_TEST_CASE_TEMPLATE_INSTANTIATE(id, ...) DOCTEST_TEST_CASE_TEMPLATE_INVOKE(id,__VA_ARGS__)
// clang-format on

// BDD style macros
// clang-format off
#define DOCTEST_SCENARIO(name) DOCTEST_TEST_CASE("  Scenario: " name)
#define DOCTEST_SCENARIO_CLASS(name) DOCTEST_TEST_CASE_CLASS("  Scenario: " name)
#define DOCTEST_SCENARIO_TEMPLATE(name, T, ...)  DOCTEST_TEST_CASE_TEMPLATE("  Scenario: " name, T, __VA_ARGS__)
#define DOCTEST_SCENARIO_TEMPLATE_DEFINE(name, T, id) DOCTEST_TEST_CASE_TEMPLATE_DEFINE("  Scenario: " name, T, id)

#define DOCTEST_GIVEN(name)     DOCTEST_SUBCASE("   Given: " name)
#define DOCTEST_WHEN(name)      DOCTEST_SUBCASE("    When: " name)
#define DOCTEST_AND_WHEN(name)  DOCTEST_SUBCASE("And when: " name)
#define DOCTEST_THEN(name)      DOCTEST_SUBCASE("    Then: " name)
#define DOCTEST_AND_THEN(name)  DOCTEST_SUBCASE("     And: " name)
// clang-format on

// == SHORT VERSIONS OF THE MACROS
#ifndef DOCTEST_CONFIG_NO_SHORT_MACRO_NAMES

#define TEST_CASE(name) DOCTEST_TEST_CASE(name)
#define TEST_CASE_CLASS(name) DOCTEST_TEST_CASE_CLASS(name)
#define TEST_CASE_FIXTURE(x, name) DOCTEST_TEST_CASE_FIXTURE(x, name)
#define TYPE_TO_STRING_AS(str, ...) DOCTEST_TYPE_TO_STRING_AS(str, __VA_ARGS__)
#define TYPE_TO_STRING(...) DOCTEST_TYPE_TO_STRING(__VA_ARGS__)
#define TEST_CASE_TEMPLATE(name, T, ...) DOCTEST_TEST_CASE_TEMPLATE(name, T, __VA_ARGS__)
#define TEST_CASE_TEMPLATE_DEFINE(name, T, id) DOCTEST_TEST_CASE_TEMPLATE_DEFINE(name, T, id)
#define TEST_CASE_TEMPLATE_INVOKE(id, ...) DOCTEST_TEST_CASE_TEMPLATE_INVOKE(id, __VA_ARGS__)
#define TEST_CASE_TEMPLATE_APPLY(id, ...) DOCTEST_TEST_CASE_TEMPLATE_APPLY(id, __VA_ARGS__)
#define SUBCASE(name) DOCTEST_SUBCASE(name)
#define TEST_SUITE(decorators) DOCTEST_TEST_SUITE(decorators)
#define TEST_SUITE_BEGIN(name) DOCTEST_TEST_SUITE_BEGIN(name)
#define TEST_SUITE_END DOCTEST_TEST_SUITE_END
#define REGISTER_EXCEPTION_TRANSLATOR(signature) DOCTEST_REGISTER_EXCEPTION_TRANSLATOR(signature)
#define REGISTER_REPORTER(name, priority, reporter) DOCTEST_REGISTER_REPORTER(name, priority, reporter)
#define REGISTER_LISTENER(name, priority, reporter) DOCTEST_REGISTER_LISTENER(name, priority, reporter)
#define INFO(...) DOCTEST_INFO(__VA_ARGS__)
#define CAPTURE(x) DOCTEST_CAPTURE(x)
```

```
#define ADD_MESSAGE_AT(file, line, ...) DOCTEST_ADD_MESSAGE_AT(file, line, __VA_ARGS__)
#define ADD_FAIL_CHECK_AT(file, line, ...) DOCTEST_ADD_FAIL_CHECK_AT(file, line, __VA_ARGS__)
#define ADD_FAIL_AT(file, line, ...) DOCTEST_ADD_FAIL_AT(file, line, __VA_ARGS__)
#define MESSAGE(...) DOCTEST_MESSAGE(__VA_ARGS__)
#define FAIL_CHECK(...) DOCTEST_FAIL_CHECK(__VA_ARGS__)
#define FAIL(...) DOCTEST_FAIL(__VA_ARGS__)
#define TO_LVALUE(...) DOCTEST_TO_LVALUE(__VA_ARGS__)

#define WARN(...) DOCTEST_WARN(__VA_ARGS__)
#define WARN_FALSE(...) DOCTEST_WARN_FALSE(__VA_ARGS__)
#define WARN_THROWS(...) DOCTEST_WARN_THROWS(__VA_ARGS__)
#define WARN_THROWS_AS(expr, ...) DOCTEST_WARN_THROWS_AS(expr, __VA_ARGS__)
#define WARN_THROWS_WITH(expr, ...) DOCTEST_WARN_THROWS_WITH(expr, __VA_ARGS__)
#define WARN_THROWS_WITH_AS(expr, with, ...) DOCTEST_WARN_THROWS_WITH_AS(expr, with, __VA_ARGS__)
#define WARN_NOTHROW(...) DOCTEST_WARN_NOTHROW(__VA_ARGS__)
#define CHECK(...) DOCTEST_CHECK(__VA_ARGS__)
#define CHECK_FALSE(...) DOCTEST_CHECK_FALSE(__VA_ARGS__)
#define CHECK_THROWS(...) DOCTEST_CHECK_THROWS(__VA_ARGS__)
#define CHECK_THROWS_AS(expr, ...) DOCTEST_CHECK_THROWS_AS(expr, __VA_ARGS__)
#define CHECK_THROWS_WITH(expr, ...) DOCTEST_CHECK_THROWS_WITH(expr, __VA_ARGS__)
#define CHECK_THROWS_WITH_AS(expr, with, ...) DOCTEST_CHECK_THROWS_WITH_AS(expr, with, __VA_ARGS__)
#define CHECK_NOTHROW(...) DOCTEST_CHECK_NOTHROW(__VA_ARGS__)
#define REQUIRE(...) DOCTEST_REQUIRE(__VA_ARGS__)
#define REQUIRE_FALSE(...) DOCTEST_REQUIRE_FALSE(__VA_ARGS__)
#define REQUIRE_THROWS(...) DOCTEST_REQUIRE_THROWS(__VA_ARGS__)
#define REQUIRE_THROWS_AS(expr, ...) DOCTEST_REQUIRE_THROWS_AS(expr, __VA_ARGS__)
#define REQUIRE_THROWS_WITH(expr, ...) DOCTEST_REQUIRE_THROWS_WITH(expr, __VA_ARGS__)
#define REQUIRE_THROWS_WITH_AS(expr, with, ...) DOCTEST_REQUIRE_THROWS_WITH_AS(expr, with,
__VA_ARGS__)
#define REQUIRE_NOTHROW(...) DOCTEST_REQUIRE_NOTHROW(__VA_ARGS__)

#define WARN_MESSAGE(cond, ...) DOCTEST_WARN_MESSAGE(cond, __VA_ARGS__)
#define WARN_FALSE_MESSAGE(cond, ...) DOCTEST_WARN_FALSE_MESSAGE(cond, __VA_ARGS__)
#define WARN_THROWS_MESSAGE(expr, ...) DOCTEST_WARN_THROWS_MESSAGE(expr, __VA_ARGS__)
#define WARN_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_WARN_THROWS_AS_MESSAGE(expr, ex, __VA_ARGS__)
#define WARN_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_WARN_THROWS_WITH_MESSAGE(expr, with,
__VA_ARGS__)
#define WARN_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_WARN_THROWS_WITH_AS_MESSAGE(expr,
with, ex, __VA_ARGS__)
#define WARN_NOTHROW_MESSAGE(expr, ...) DOCTEST_WARN_NOTHROW_MESSAGE(expr, __VA_ARGS__)
#define CHECK_MESSAGE(cond, ...) DOCTEST_CHECK_MESSAGE(cond, __VA_ARGS__)
#define CHECK_FALSE_MESSAGE(cond, ...) DOCTEST_CHECK_FALSE_MESSAGE(cond, __VA_ARGS__)
#define CHECK_THROWS_MESSAGE(expr, ...) DOCTEST_CHECK_THROWS_MESSAGE(expr, __VA_ARGS__)
#define CHECK_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_CHECK_THROWS_AS_MESSAGE(expr, ex, __VA_ARGS__)
#define CHECK_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_CHECK_THROWS_WITH_MESSAGE(expr, with,
__VA_ARGS__)
#define CHECK_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_CHECK_THROWS_WITH_AS_MESSAGE(expr,
with, ex, __VA_ARGS__)
#define CHECK_NOTHROW_MESSAGE(expr, ...) DOCTEST_CHECK_NOTHROW_MESSAGE(expr, __VA_ARGS__)
#define REQUIRE_MESSAGE(cond, ...) DOCTEST_REQUIRE_MESSAGE(cond, __VA_ARGS__)
#define REQUIRE_FALSE_MESSAGE(cond, ...) DOCTEST_REQUIRE_FALSE_MESSAGE(cond, __VA_ARGS__)
#define REQUIRE_THROWS_MESSAGE(expr, ...) DOCTEST_REQUIRE_THROWS_MESSAGE(expr, __VA_ARGS__)
#define REQUIRE_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_REQUIRE_THROWS_AS_MESSAGE(expr, ex,
__VA_ARGS__)
#define REQUIRE_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_REQUIRE_THROWS_WITH_MESSAGE(expr, with,
__VA_ARGS__)
#define REQUIRE_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...)
DOCTEST_REQUIRE_THROWS_WITH_AS_MESSAGE(expr, with, ex, __VA_ARGS__)
#define REQUIRE_NOTHROW_MESSAGE(expr, ...) DOCTEST_REQUIRE_NOTHROW_MESSAGE(expr, __VA_ARGS__)

#define SCENARIO(name) DOCTEST_SCENARIO(name)
#define SCENARIO_CLASS(name) DOCTEST_SCENARIO_CLASS(name)
#define SCENARIO_TEMPLATE(name, T, ...) DOCTEST_SCENARIO_TEMPLATE(name, T, __VA_ARGS__)
#define SCENARIO_TEMPLATE_DEFINE(name, T, id) DOCTEST_SCENARIO_TEMPLATE_DEFINE(name, T, id)
#define GIVEN(name) DOCTEST_GIVEN(name)
#define WHEN(name) DOCTEST_WHEN(name)
#define AND_WHEN(name) DOCTEST_AND_WHEN(name)
#define THEN(name) DOCTEST_THEN(name)
#define AND_THEN(name) DOCTEST_AND_THEN(name)

#define WARN_EQ(...) DOCTEST_WARN_EQ(__VA_ARGS__)
#define CHECK_EQ(...) DOCTEST_CHECK_EQ(__VA_ARGS__)
#define REQUIRE_EQ(...) DOCTEST_REQUIRE_EQ(__VA_ARGS__)
#define WARN_NE(...) DOCTEST_WARN_NE(__VA_ARGS__)
#define CHECK_NE(...) DOCTEST_CHECK_NE(__VA_ARGS__)
#define REQUIRE_NE(...) DOCTEST_REQUIRE_NE(__VA_ARGS__)
#define WARN_GT(...) DOCTEST_WARN_GT(__VA_ARGS__)
#define CHECK_GT(...) DOCTEST_CHECK_GT(__VA_ARGS__)
#define REQUIRE_GT(...) DOCTEST_REQUIRE_GT(__VA_ARGS__)
#define WARN_LT(...) DOCTEST_WARN_LT(__VA_ARGS__)
#define CHECK_LT(...) DOCTEST_CHECK_LT(__VA_ARGS__)
#define REQUIRE_LT(...) DOCTEST_REQUIRE_LT(__VA_ARGS__)
#define WARN_GE(...) DOCTEST_WARN_GE(__VA_ARGS__)
#define CHECK_GE(...) DOCTEST_CHECK_GE(__VA_ARGS__)
#define REQUIRE_GE(...) DOCTEST_REQUIRE_GE(__VA_ARGS__)
#define WARN_LE(...) DOCTEST_WARN_LE(__VA_ARGS__)
#define CHECK_LE(...) DOCTEST_CHECK_LE(__VA_ARGS__)
```

```
#define REQUIRE_LE(...) DOCTEST_REQUIRE_LE(__VA_ARGS__)
#define WARN_UNARY(...) DOCTEST_WARN_UNARY(__VA_ARGS__)
#define CHECK_UNARY(...) DOCTEST_CHECK_UNARY(__VA_ARGS__)
#define REQUIRE_UNARY(...) DOCTEST_REQUIRE_UNARY(__VA_ARGS__)
#define WARN_UNARY_FALSE(...) DOCTEST_WARN_UNARY_FALSE(__VA_ARGS__)
#define CHECK_UNARY_FALSE(...) DOCTEST_CHECK_UNARY_FALSE(__VA_ARGS__)
#define REQUIRE_UNARY_FALSE(...) DOCTEST_REQUIRE_UNARY_FALSE(__VA_ARGS__)

// KEPT FOR BACKWARDS COMPATIBILITY
#define FAST_WARN_EQ(...) DOCTEST_FAST_WARN_EQ(__VA_ARGS__)
#define FAST_CHECK_EQ(...) DOCTEST_FAST_CHECK_EQ(__VA_ARGS__)
#define FAST_REQUIRE_EQ(...) DOCTEST_FAST_REQUIRE_EQ(__VA_ARGS__)
#define FAST_WARN_NE(...) DOCTEST_FAST_WARN_NE(__VA_ARGS__)
#define FAST_CHECK_NE(...) DOCTEST_FAST_CHECK_NE(__VA_ARGS__)
#define FAST_REQUIRE_NE(...) DOCTEST_FAST_REQUIRE_NE(__VA_ARGS__)
#define FAST_WARN_GT(...) DOCTEST_FAST_WARN_GT(__VA_ARGS__)
#define FAST_CHECK_GT(...) DOCTEST_FAST_CHECK_GT(__VA_ARGS__)
#define FAST_REQUIRE_GT(...) DOCTEST_FAST_REQUIRE_GT(__VA_ARGS__)
#define FAST_WARN_LT(...) DOCTEST_FAST_WARN_LT(__VA_ARGS__)
#define FAST_CHECK_LT(...) DOCTEST_FAST_CHECK_LT(__VA_ARGS__)
#define FAST_REQUIRE_LT(...) DOCTEST_FAST_REQUIRE_LT(__VA_ARGS__)
#define FAST_WARN_GE(...) DOCTEST_FAST_WARN_GE(__VA_ARGS__)
#define FAST_CHECK_GE(...) DOCTEST_FAST_CHECK_GE(__VA_ARGS__)
#define FAST_REQUIRE_GE(...) DOCTEST_FAST_REQUIRE_GE(__VA_ARGS__)
#define FAST_WARN_LE(...) DOCTEST_FAST_WARN_LE(__VA_ARGS__)
#define FAST_CHECK_LE(...) DOCTEST_FAST_CHECK_LE(__VA_ARGS__)
#define FAST_REQUIRE_LE(...) DOCTEST_FAST_REQUIRE_LE(__VA_ARGS__)

#define FAST_WARN_UNARY(...) DOCTEST_FAST_WARN_UNARY(__VA_ARGS__)
#define FAST_CHECK_UNARY(...) DOCTEST_FAST_CHECK_UNARY(__VA_ARGS__)
#define FAST_REQUIRE_UNARY(...) DOCTEST_FAST_REQUIRE_UNARY(__VA_ARGS__)
#define FAST_WARN_UNARY_FALSE(...) DOCTEST_FAST_WARN_UNARY_FALSE(__VA_ARGS__)
#define FAST_CHECK_UNARY_FALSE(...) DOCTEST_FAST_CHECK_UNARY_FALSE(__VA_ARGS__)
#define FAST_REQUIRE_UNARY_FALSE(...) DOCTEST_FAST_REQUIRE_UNARY_FALSE(__VA_ARGS__)

#define TEST_CASE_TEMPLATE_INSTANTIATE(id, ...) DOCTEST_TEST_CASE_TEMPLATE_INSTANTIATE(id,
    __VA_ARGS__)

#endif // DOCTEST_CONFIG_NO_SHORT_MACRO_NAMES

#ifndef DOCTEST_CONFIG_DISABLE

// this is here to clear the 'current test suite' for the current translation unit - at the top
DOCTEST_TEST_SUITE_END();

#endif // DOCTEST_CONFIG_DISABLE

DOCTEST_CLANG_SUPPRESS_WARNING_POP
DOCTEST_MSVC_SUPPRESS_WARNING_POP
DOCTEST_GCC_SUPPRESS_WARNING_POP

DOCTEST_SUPPRESS_COMMON_WARNINGS_POP

#endif // DOCTEST_LIBRARY_INCLUDED

#ifndef DOCTEST_SINGLE_HEADER
#define DOCTEST_SINGLE_HEADER
#endif // DOCTEST_SINGLE_HEADER

#if defined(DOCTEST_CONFIG_IMPLEMENT) || !defined(DOCTEST_SINGLE_HEADER)

#ifndef DOCTEST_SINGLE_HEADER
#include "doctest_fwd.h"
#endif // DOCTEST_SINGLE_HEADER

DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wunused-macros")

#ifndef DOCTEST_LIBRARY_IMPLEMENTATION
#define DOCTEST_LIBRARY_IMPLEMENTATION

DOCTEST_CLANG_SUPPRESS_WARNING_POP

DOCTEST_SUPPRESS_COMMON_WARNINGS_PUSH

DOCTEST_CLANG_SUPPRESS_WARNING_PUSH
DOCTEST_CLANG_SUPPRESS_WARNING("-Wglobal-constructors")
DOCTEST_CLANG_SUPPRESS_WARNING("-Wexit-time-destructors")
DOCTEST_CLANG_SUPPRESS_WARNING("-Wsign-conversion")
DOCTEST_CLANG_SUPPRESS_WARNING("-Wshorten-64-to-32")
DOCTEST_CLANG_SUPPRESS_WARNING("-Wmissing-variable-declarations")
DOCTEST_CLANG_SUPPRESS_WARNING("-Wswitch")
DOCTEST_CLANG_SUPPRESS_WARNING("-Wswitch-enum")
DOCTEST_CLANG_SUPPRESS_WARNING("-Wcovered-switch-default")
DOCTEST_CLANG_SUPPRESS_WARNING("-Wmissing-noreturn")
DOCTEST_CLANG_SUPPRESS_WARNING("-Wdisabled-macro-expansion")
DOCTEST_CLANG_SUPPRESS_WARNING("-Wmissing-braces")
DOCTEST_CLANG_SUPPRESS_WARNING("-Wmissing-field-initializers")
DOCTEST_CLANG_SUPPRESS_WARNING("-Wunused-member-function")
DOCTEST_CLANG_SUPPRESS_WARNING("-Wnonportable-system-include-path")

DOCTEST_GCC_SUPPRESS_WARNING_PUSH
DOCTEST_GCC_SUPPRESS_WARNING("-Wconversion")
DOCTEST_GCC_SUPPRESS_WARNING("-Wsign-conversion")
DOCTEST_GCC_SUPPRESS_WARNING("-Wmissing-field-initializers")
DOCTEST_GCC_SUPPRESS_WARNING("-Wmissing-braces")
DOCTEST_GCC_SUPPRESS_WARNING("-Wswitch")
DOCTEST_GCC_SUPPRESS_WARNING("-Wswitch-enum")
DOCTEST_GCC_SUPPRESS_WARNING("-Wswitch-default")
DOCTEST_GCC_SUPPRESS_WARNING("-Wunsafe-loop-optimizations")
DOCTEST_GCC_SUPPRESS_WARNING("-Wold-style-cast")
DOCTEST_GCC_SUPPRESS_WARNING("-Wunused-function")
DOCTEST_GCC_SUPPRESS_WARNING("-Wmultiple-inheritance")
DOCTEST_GCC_SUPPRESS_WARNING("-Wsuggest-attribute")

DOCTEST_MSVC_SUPPRESS_WARNING_PUSH
DOCTEST_MSVC_SUPPRESS_WARNING(4267) // 'var' : conversion from 'x' to 'y', possible loss of data
DOCTEST_MSVC_SUPPRESS_WARNING(4530) // C++ exception handler used, but unwind semantics not enabled
DOCTEST_MSVC_SUPPRESS_WARNING(4577) // 'noexcept' used with no exception handling mode specified
DOCTEST_MSVC_SUPPRESS_WARNING(4774) // format string expected in argument is not a string literal
DOCTEST_MSVC_SUPPRESS_WARNING(4365) // conversion from 'int' to 'unsigned', signed/unsigned mismatch
DOCTEST_MSVC_SUPPRESS_WARNING(5039) // pointer to potentially throwing function passed to extern C
DOCTEST_MSVC_SUPPRESS_WARNING(4800) // forcing value to bool 'true' or 'false' (performance warning)
DOCTEST_MSVC_SUPPRESS_WARNING(5245) // unreferenced function with internal linkage has been removed

DOCTEST_MAKE_STD_HEADERS_CLEAN_FROM_WARNINGS_ON_WALL_BEGIN

// required includes - will go only in one translation unit!
#include <ctime>
#include <cmath>
#include <climits>
// borland (Embarcadero) compiler requires math.h and not cmath -
https://github.com/doctest/doctest/pull/37
#ifdef __BORLANDC__
#include <math.h>
#endif // __BORLANDC__
#include <new>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <limits>
#include <utility>
#include <fstream>
#include <sstream>
#ifndef DOCTEST_CONFIG_NO_INCLUDE_IOSTREAM
#include <iostream>
#endif // DOCTEST_CONFIG_NO_INCLUDE_IOSTREAM
#include <algorithm>
#include <iomanip>
#include <vector>
#ifndef DOCTEST_CONFIG_NO_MULTITHREADING
#include <atomic>
#include <mutex>
#define DOCTEST_DECLARE_MUTEX(name) std::mutex name;
#define DOCTEST_DECLARE_STATIC_MUTEX(name) static DOCTEST_DECLARE_MUTEX(name)
#define DOCTEST_LOCK_MUTEX(name) std::lock_guard<std::mutex>
DOCTEST_ANONYMOUS(DOCTEST_ANON_LOCK_)(name);
#else // DOCTEST_CONFIG_NO_MULTITHREADING
#define DOCTEST_DECLARE_MUTEX(name)
#define DOCTEST_DECLARE_STATIC_MUTEX(name)
#define DOCTEST_LOCK_MUTEX(name)
#endif // DOCTEST_CONFIG_NO_MULTITHREADING
#include <set>
#include <map>
#include <unordered_set>
#include <exception>
#include <stdexcept>
#include <csignal>
#include <cfloat>
#include <cctype>
#include <cstdint>
#include <string>

#ifdef DOCTEST_PLATFORM_MAC
#include <sys/types.h>
#include <unistd.h>
#include <sys/sysctl.h>
#endif // DOCTEST_PLATFORM_MAC

#ifdef DOCTEST_PLATFORM_WINDOWS

// defines for a leaner windows.h
#ifndef WIN32_LEAN_AND_MEAN
#define WIN32_LEAN_AND_MEAN
#define DOCTEST_UNDEF_WIN32_LEAN_AND_MEAN
```

```
#endif // WIN32_LEAN_AND_MEAN
#ifndef NOMINMAX
#define NOMINMAX
#define DOCTEST_UNDEF_NOMINMAX
#endif // NOMINMAX

// not sure what AfxWin.h is for - here I do what Catch does
#ifdef __AFXDLL
#include <AfxWin.h>
#else
#include <windows.h>
#endif
#include <io.h>

#else // DOCTEST_PLATFORM_WINDOWS

#include <sys/time.h>
#include <unistd.h>

#endif // DOCTEST_PLATFORM_WINDOWS

// this is a fix for https://github.com/doctest/doctest/issues/348
// https://mail.gnome.org/archives/xml/2012-January/msg00000.html
#if !defined(HAVE_UNISTD_H) && !defined(STDOUT_FILENO)
#define STDOUT_FILENO fileno(stdout)
#endif // HAVE_UNISTD_H

DOCTEST_MAKE_STD_HEADERS_CLEAN_FROM_WARNINGS_ON_WALL_END

// counts the number of elements in a C array
#define DOCTEST_COUNTOF(x) (sizeof(x) / sizeof(x[0]))

#ifdef DOCTEST_CONFIG_DISABLE
#define DOCTEST_BRANCH_ON_DISABLED(if_disabled, if_not_disabled) if_disabled
#else // DOCTEST_CONFIG_DISABLE
#define DOCTEST_BRANCH_ON_DISABLED(if_disabled, if_not_disabled) if_not_disabled
#endif // DOCTEST_CONFIG_DISABLE

#ifndef DOCTEST_CONFIG_OPTIONS_PREFIX
#define DOCTEST_CONFIG_OPTIONS_PREFIX "dt-"
#endif

#ifndef DOCTEST_CONFIG_OPTIONS_FILE_PREFIX_SEPARATOR
#define DOCTEST_CONFIG_OPTIONS_FILE_PREFIX_SEPARATOR ':'
#endif

#ifndef DOCTEST_THREAD_LOCAL
#if defined(DOCTEST_CONFIG_NO_MULTITHREADING) || DOCTEST_MSVC && (DOCTEST_MSVC < DOCTEST_COMPILER(19,
    0, 0))
#define DOCTEST_THREAD_LOCAL
#else // DOCTEST_MSVC
#define DOCTEST_THREAD_LOCAL thread_local
#endif // DOCTEST_MSVC
#endif // DOCTEST_THREAD_LOCAL

#ifndef DOCTEST_MULTI_LANE_ATOMICS_THREAD_LANES
#define DOCTEST_MULTI_LANE_ATOMICS_THREAD_LANES 32
#endif

#ifndef DOCTEST_MULTI_LANE_ATOMICS_CACHE_LINE_SIZE
#define DOCTEST_MULTI_LANE_ATOMICS_CACHE_LINE_SIZE 64
#endif

#ifdef DOCTEST_CONFIG_NO_UNPREFIXED_OPTIONS
#define DOCTEST_OPTIONS_PREFIX_DISPLAY DOCTEST_CONFIG_OPTIONS_PREFIX
#else
#define DOCTEST_OPTIONS_PREFIX_DISPLAY ""
#endif

#if defined(WINAPI_FAMILY) && (WINAPI_FAMILY == WINAPI_FAMILY_APP)
#define DOCTEST_CONFIG_NO_MULTI_LANE_ATOMICS
#endif

#ifndef DOCTEST_CDECL
#define DOCTEST_CDECL __cdecl
#endif

namespace doctest {

bool is_running_in_test = false;

namespace {
    using namespace detail;

    template <typename Ex>
    DOCTEST_NORETURN void throw_exception(Ex const& e) {
#ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
```

```
        throw e;
#else  // DOCTEST_CONFIG_NO_EXCEPTIONS
#ifdef DOCTEST_CONFIG_HANDLE_EXCEPTION
        DOCTEST_CONFIG_HANDLE_EXCEPTION(e);
#else // DOCTEST_CONFIG_HANDLE_EXCEPTION
#ifndef DOCTEST_CONFIG_NO_INCLUDE_IOSTREAM
        std::cerr « "doctest will terminate because it needed to throw an exception.\n"
                  « "The message was: " « e.what() « '\n';
#endif // DOCTEST_CONFIG_NO_INCLUDE_IOSTREAM
#endif // DOCTEST_CONFIG_HANDLE_EXCEPTION
        std::terminate();
#endif // DOCTEST_CONFIG_NO_EXCEPTIONS
    }

#ifndef DOCTEST_INTERNAL_ERROR
#define DOCTEST_INTERNAL_ERROR(msg)                                                                \
    throw_exception(std::logic_error(                                                              \
            __FILE__ ":" DOCTEST_TOSTR(__LINE__) ": Internal doctest error: " msg))
#endif // DOCTEST_INTERNAL_ERROR

    // case insensitive strcmp
    int stricmp(const char* a, const char* b) {
        for(;; a++, b++) {
            const int d = tolower(*a) - tolower(*b);
            if(d != 0 || !*a)
                return d;
        }
    }

    struct Endianness
    {
        enum Arch
        {
            Big,
            Little
        };

        static Arch which() {
            int x = 1;
            // casting any data pointer to char* is allowed
            auto ptr = reinterpret_cast<char*>(&x);
            if(*ptr)
                return Little;
            return Big;
        }
    };
} // namespace

namespace detail {
    DOCTEST_THREAD_LOCAL class
    {
        std::vector<std::streampos> stack;
        std::stringstream           ss;

    public:
        std::ostream* push() {
            stack.push_back(ss.tellp());
            return &ss;
        }

        String pop() {
            if (stack.empty())
                DOCTEST_INTERNAL_ERROR("TLSS was empty when trying to pop!");

            std::streampos pos = stack.back();
            stack.pop_back();
            unsigned sz = static_cast<unsigned>(ss.tellp() - pos);
            ss.rdbuf()->pubseekpos(pos, std::ios::in | std::ios::out);
            return String(ss, sz);
        }
    } g_oss;

    std::ostream* tlssPush() {
        return g_oss.push();
    }

    String tlssPop() {
        return g_oss.pop();
    }

#ifndef DOCTEST_CONFIG_DISABLE

namespace timer_large_integer
{

#if defined(DOCTEST_PLATFORM_WINDOWS)
    using type = ULONGLONG;
```

```
#else // DOCTEST_PLATFORM_WINDOWS
    using type = std::uint64_t;
#endif // DOCTEST_PLATFORM_WINDOWS
}

using ticks_t = timer_large_integer::type;

#ifdef DOCTEST_CONFIG_GETCURRENTTICKS
    ticks_t getCurrentTicks() { return DOCTEST_CONFIG_GETCURRENTTICKS(); }
#elif defined(DOCTEST_PLATFORM_WINDOWS)
    ticks_t getCurrentTicks() {
        static LARGE_INTEGER hz = { {0} }, hzo = { {0} };
        if(!hz.QuadPart) {
            QueryPerformanceFrequency(&hz);
            QueryPerformanceCounter(&hzo);
        }
        LARGE_INTEGER t;
        QueryPerformanceCounter(&t);
        return ((t.QuadPart - hzo.QuadPart) * LONGLONG(1000000)) / hz.QuadPart;
    }
#else  // DOCTEST_PLATFORM_WINDOWS
    ticks_t getCurrentTicks() {
        timeval t;
        gettimeofday(&t, nullptr);
        return static_cast<ticks_t>(t.tv_sec) * 1000000 + static_cast<ticks_t>(t.tv_usec);
    }
#endif // DOCTEST_PLATFORM_WINDOWS

    struct Timer
    {
        void         start() { m_ticks = getCurrentTicks(); }
        unsigned int getElapsedMicroseconds() const {
            return static_cast<unsigned int>(getCurrentTicks() - m_ticks);
        }
        //unsigned int getElapsedMilliseconds() const {
        //    return static_cast<unsigned int>(getElapsedMicroseconds() / 1000);
        //}
        double getElapsedSeconds() const { return static_cast<double>(getCurrentTicks() - m_ticks) /
1000000.0; }

    private:
        ticks_t m_ticks = 0;
    };

#ifdef DOCTEST_CONFIG_NO_MULTITHREADING
    template <typename T>
    using Atomic = T;
#else // DOCTEST_CONFIG_NO_MULTITHREADING
    template <typename T>
    using Atomic = std::atomic<T>;
#endif // DOCTEST_CONFIG_NO_MULTITHREADING

#if defined(DOCTEST_CONFIG_NO_MULTI_LANE_ATOMICS) || defined(DOCTEST_CONFIG_NO_MULTITHREADING)
    template <typename T>
    using MultiLaneAtomic = Atomic<T>;
#else // DOCTEST_CONFIG_NO_MULTI_LANE_ATOMICS
    // Provides a multilane implementation of an atomic variable that supports add, sub, load,
    // store. Instead of using a single atomic variable, this splits up into multiple ones,
    // each sitting on a separate cache line. The goal is to provide a speedup when most
    // operations are modifying. It achieves this with two properties:
    //
    // * Multiple atomics are used, so chance of congestion from the same atomic is reduced.
    // * Each atomic sits on a separate cache line, so false sharing is reduced.
    //
    // The disadvantage is that there is a small overhead due to the use of TLS, and load/store
    // is slower because all atomics have to be accessed.
    template <typename T>
    class MultiLaneAtomic
    {
        struct CacheLineAlignedAtomic
        {
            Atomic<T> atomic{};
            char padding[DOCTEST_MULTI_LANE_ATOMICS_CACHE_LINE_SIZE - sizeof(Atomic<T>)];
        };
        CacheLineAlignedAtomic m_atomics[DOCTEST_MULTI_LANE_ATOMICS_THREAD_LANES];

        static_assert(sizeof(CacheLineAlignedAtomic) == DOCTEST_MULTI_LANE_ATOMICS_CACHE_LINE_SIZE,
                      "guarantee one atomic takes exactly one cache line");

    public:
        T operator++() DOCTEST_NOEXCEPT { return fetch_add(1) + 1; }

        T operator++(int) DOCTEST_NOEXCEPT { return fetch_add(1); }

        T fetch_add(T arg, std::memory_order order = std::memory_order_seq_cst) DOCTEST_NOEXCEPT {
            return myAtomic().fetch_add(arg, order);
        }

        T fetch_sub(T arg, std::memory_order order = std::memory_order_seq_cst) DOCTEST_NOEXCEPT {
            return myAtomic().fetch_sub(arg, order);
        }

        operator T() const DOCTEST_NOEXCEPT { return load(); }

        T load(std::memory_order order = std::memory_order_seq_cst) const DOCTEST_NOEXCEPT {
            auto result = T();
            for(auto const& c : m_atomics) {
                result += c.atomic.load(order);
            }
            return result;
        }

        T operator=(T desired) DOCTEST_NOEXCEPT { // lgtm [cpp/assignment-does-not-return-this]
            store(desired);
            return desired;
        }

        void store(T desired, std::memory_order order = std::memory_order_seq_cst) DOCTEST_NOEXCEPT {
            // first value becomes desired", all others become 0.
            for(auto& c : m_atomics) {
                c.atomic.store(desired, order);
                desired = {};
            }
        }

    private:
        // Each thread has a different atomic that it operates on. If more than NumLanes threads
        // use this, some will use the same atomic. So performance will degrade a bit, but still
        // everything will work.
        //
        // The logic here is a bit tricky. The call should be as fast as possible, so that there
        // is minimal to no overhead in determining the correct atomic for the current thread.
        //
        // 1. A global static counter laneCounter counts continuously up.
        // 2. Each successive thread will use modulo operation of that counter so it gets an atomic
        //    assigned in a round-robin fashion.
        // 3. This tlsLaneIdx is stored in the thread local data, so it is directly available with
        //    little overhead.
        Atomic<T>& myAtomic() DOCTEST_NOEXCEPT {
            static Atomic<size_t> laneCounter;
            DOCTEST_THREAD_LOCAL size_t tlsLaneIdx =
                    laneCounter++ % DOCTEST_MULTI_LANE_ATOMICS_THREAD_LANES;

            return m_atomics[tlsLaneIdx].atomic;
        }
    };
#endif // DOCTEST_CONFIG_NO_MULTI_LANE_ATOMICS

    // this holds both parameters from the command line and runtime data for tests
    struct ContextState : ContextOptions, TestRunStats, CurrentTestCaseStats
    {
        MultiLaneAtomic<int> numAssertsCurrentTest_atomic;
        MultiLaneAtomic<int> numAssertsFailedCurrentTest_atomic;

        std::vector<std::vector<String» filters = decltype(filters)(9); // 9 different filters

        std::vector<IReporter*> reporters_currently_used;

        assert_handler ah = nullptr;

        Timer timer;

        std::vector<String> stringifiedContexts; // logging from INFO() due to an exception

        // stuff for subcases
        bool reachedLeaf;
        std::vector<SubcaseSignature> subcaseStack;
        std::vector<SubcaseSignature> nextSubcaseStack;
        std::unordered_set<unsigned long long> fullyTraversedSubcases;
        size_t currentSubcaseDepth;
        Atomic<bool> shouldLogCurrentException;

        void resetRunData() {
            numTestCases                = 0;
            numTestCasesPassingFilters  = 0;
            numTestSuitesPassingFilters = 0;
            numTestCasesFailed          = 0;
            numAsserts                  = 0;
            numAssertsFailed            = 0;
            numAssertsCurrentTest       = 0;
            numAssertsFailedCurrentTest = 0;
        }

        void finalizeTestCaseData() {
            seconds = timer.getElapsedSeconds();

            // update the non-atomic counters
            numAsserts += numAssertsCurrentTest_atomic;
            numAssertsFailed += numAssertsFailedCurrentTest_atomic;
            numAssertsCurrentTest       = numAssertsCurrentTest_atomic;
            numAssertsFailedCurrentTest = numAssertsFailedCurrentTest_atomic;

            if(numAssertsFailedCurrentTest)
                failure_flags |= TestCaseFailureReason::AssertFailure;

            if(Approx(currentTest->m_timeout).epsilon(DBL_EPSILON) != 0 &&
               Approx(seconds).epsilon(DBL_EPSILON) > currentTest->m_timeout)
                failure_flags |= TestCaseFailureReason::Timeout;

            if(currentTest->m_should_fail) {
                if(failure_flags) {
                    failure_flags |= TestCaseFailureReason::ShouldHaveFailedAndDid;
                } else {
                    failure_flags |= TestCaseFailureReason::ShouldHaveFailedButDidnt;
                }
            } else if(failure_flags && currentTest->m_may_fail) {
                failure_flags |= TestCaseFailureReason::CouldHaveFailedAndDid;
            } else if(currentTest->m_expected_failures > 0) {
                if(numAssertsFailedCurrentTest == currentTest->m_expected_failures) {
                    failure_flags |= TestCaseFailureReason::FailedExactlyNumTimes;
                } else {
                    failure_flags |= TestCaseFailureReason::DidntFailExactlyNumTimes;
                }
            }

            bool ok_to_fail = (TestCaseFailureReason::ShouldHaveFailedAndDid & failure_flags) ||
                              (TestCaseFailureReason::CouldHaveFailedAndDid & failure_flags) ||
                              (TestCaseFailureReason::FailedExactlyNumTimes & failure_flags);

            // if any subcase has failed - the whole test case has failed
            testCaseSuccess = !(failure_flags && !ok_to_fail);
            if(!testCaseSuccess)
                numTestCasesFailed++;
        }
    };

    ContextState* g_cs = nullptr;

    // used to avoid locks for the debug output
    // TODO: figure out if this is indeed necessary/correct - seems like either there still
    // could be a race or that there wouldn't be a race even if using the context directly
    DOCTEST_THREAD_LOCAL bool g_no_colors;

#endif // DOCTEST_CONFIG_DISABLE
} // namespace detail

char* String::allocate(size_type sz) {
    if (sz <= last) {
        buf[sz] = '\0';
        setLast(last - sz);
        return buf;
    } else {
        setOnHeap();
        data.size = sz;
        data.capacity = data.size + 1;
        data.ptr = new char[data.capacity];
        data.ptr[sz] = '\0';
        return data.ptr;
    }
}

void String::setOnHeap() noexcept { *reinterpret_cast<unsigned char*>(&buf[last]) = 128; }
void String::setLast(size_type in) noexcept { buf[last] = char(in); }
void String::setSize(size_type sz) noexcept {
    if (isOnStack()) { buf[sz] = '\0'; setLast(last - sz); }
    else { data.ptr[sz] = '\0'; data.size = sz; }
}

void String::copy(const String& other) {
    if(other.isOnStack()) {
        memcpy(buf, other.buf, len);
    } else {
        memcpy(allocate(other.data.size), other.data.ptr, other.data.size);
    }
}

String::String() noexcept {
    buf[0] = '\0';
    setLast();
}

```

```
String::~String() {
    if(!isOnStack())
        delete[] data.ptr;
} // NOLINT(clang-analyzer-cplusplus.NewDeleteLeaks)

String::String(const char* in)
        : String(in, strlen(in)) {}

String::String(const char* in, size_type in_size) {
    memcpy(allocate(in_size), in, in_size);
}

String::String(std::istream& in, size_type in_size) {
    in.read(allocate(in_size), in_size);
}

String::String(const String& other) { copy(other); }

String& String::operator=(const String& other) {
    if(this != &other) {
        if(!isOnStack())
            delete[] data.ptr;

        copy(other);
    }

    return *this;
}

String& String::operator+=(const String& other) {
    const size_type my_old_size = size();
    const size_type other_size  = other.size();
    const size_type total_size  = my_old_size + other_size;
    if(isOnStack()) {
        if(total_size < len) {
            // append to the current stack space
            memcpy(buf + my_old_size, other.c_str(), other_size + 1);
            // NOLINTNEXTLINE(clang-analyzer-cplusplus.NewDeleteLeaks)
            setLast(last - total_size);
        } else {
            // alloc new chunk
            char* temp = new char[total_size + 1];
            // copy current data to new location before writing in the union
            memcpy(temp, buf, my_old_size); // skip the +1 ('\0') for speed
            // update data in union
            setOnHeap();
            data.size     = total_size;
            data.capacity = data.size + 1;
            data.ptr      = temp;
            // transfer the rest of the data
            memcpy(data.ptr + my_old_size, other.c_str(), other_size + 1);
        }
    } else {
        if(data.capacity > total_size) {
            // append to the current heap block
            data.size = total_size;
            memcpy(data.ptr + my_old_size, other.c_str(), other_size + 1);
        } else {
            // resize
            data.capacity *= 2;
            if(data.capacity <= total_size)
                data.capacity = total_size + 1;
            // alloc new chunk
            char* temp = new char[data.capacity];
            // copy current data to new location before releasing it
            memcpy(temp, data.ptr, my_old_size); // skip the +1 ('\0') for speed
            // release old chunk
            delete[] data.ptr;
            // update the rest of the union members
            data.size = total_size;
            data.ptr  = temp;
            // transfer the rest of the data
            memcpy(data.ptr + my_old_size, other.c_str(), other_size + 1);
        }
    }

    return *this;
}

String::String(String&& other) noexcept {
    memcpy(buf, other.buf, len);
    other.buf[0] = '\0';
    other.setLast();
}

String& String::operator=(String&& other) noexcept {
    if(this != &other) {
        if(!isOnStack())
            delete[] data.ptr;
        memcpy(buf, other.buf, len);
        other.buf[0] = ’\0’;
        other.setLast();
    }
    return *this;
}

char String::operator[](size_type i) const {
    return const_cast<String*>(this)->operator[](i);
}

char& String::operator[](size_type i) {
    if(isOnStack())
        return reinterpret_cast<char*>(buf)[i];
    return data.ptr[i];
}

DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH("-Wmaybe-uninitialized")
String::size_type String::size() const {
    if(isOnStack())
        return last - (size_type(buf[last]) & 31); // using "last" would work only if "len" is 32
    return data.size;
}
DOCTEST_GCC_SUPPRESS_WARNING_POP

String::size_type String::capacity() const {
    if(isOnStack())
        return len;
    return data.capacity;
}

String String::substr(size_type pos, size_type cnt) && {
    cnt = std::min(cnt, size() - pos);
    char* cptr = c_str();
    memmove(cptr, cptr + pos, cnt);
    setSize(cnt);
    return std::move(*this);
}

String String::substr(size_type pos, size_type cnt) const & {
    cnt = std::min(cnt, size() - pos);
    return String{ c_str() + pos, cnt };
}

String::size_type String::find(char ch, size_type pos) const {
    const char* begin = c_str();
    const char* end = begin + size();
    const char* it = begin + pos;
    for (; it < end && *it != ch; it++);
    if (it < end) { return static_cast<size_type>(it - begin); }
    else { return npos; }
}

String::size_type String::rfind(char ch, size_type pos) const {
    const char* begin = c_str();
    const char* it = begin + std::min(pos, size() - 1);
    for (; it >= begin && *it != ch; it--);
    if (it >= begin) { return static_cast<size_type>(it - begin); }
    else { return npos; }
}

int String::compare(const char* other, bool no_case) const {
    if(no_case)
        return doctest::stricmp(c_str(), other);
    return std::strcmp(c_str(), other);
}

int String::compare(const String& other, bool no_case) const {
    return compare(other.c_str(), no_case);
}

String operator+(const String& lhs, const String& rhs) { return  String(lhs) += rhs; }

bool operator==(const String& lhs, const String& rhs) { return lhs.compare(rhs) == 0; }
bool operator!=(const String& lhs, const String& rhs) { return lhs.compare(rhs) != 0; }
bool operator< (const String& lhs, const String& rhs) { return lhs.compare(rhs) < 0; }
bool operator> (const String& lhs, const String& rhs) { return lhs.compare(rhs) > 0; }
bool operator<=(const String& lhs, const String& rhs) { return (lhs != rhs) ? lhs.compare(rhs) < 0 : true; }
bool operator>=(const String& lhs, const String& rhs) { return (lhs != rhs) ? lhs.compare(rhs) > 0 : true; }

std::ostream& operator«(std::ostream& s, const String& in) { return s « in.c_str(); }

Contains::Contains(const String& str) : string(str) { }
```

```

bool Contains::checkWith(const String& other) const {
    return strstr(other.c_str(), string.c_str()) != nullptr;
}

String toString(const Contains& in) {
    return "Contains( " + in.string + " )";
}

bool operator==(const String& lhs, const Contains& rhs) { return rhs.checkWith(lhs); }
bool operator==(const Contains& lhs, const String& rhs) { return lhs.checkWith(rhs); }
bool operator!=(const String& lhs, const Contains& rhs) { return !rhs.checkWith(lhs); }
bool operator!=(const Contains& lhs, const String& rhs) { return !lhs.checkWith(rhs); }

namespace {
    void color_to_stream(std::ostream&, Color::Enum) DOCTEST_BRANCH_ON_DISABLED({}, ;)
} // namespace

namespace Color {
    std::ostream& operator«(std::ostream& s, Color::Enum code) {
        color_to_stream(s, code);
        return s;
    }
} // namespace Color

// clang-format off
const char* assertString(assertType::Enum at) {
    DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4061) // enum 'x' in switch of enum 'y' is not explicitly
handled
    #define DOCTEST_GENERATE_ASSERT_TYPE_CASE(assert_type) case assertType::DT_ ## assert_type: return
#assert_type
    #define DOCTEST_GENERATE_ASSERT_TYPE_CASES(assert_type) \
        DOCTEST_GENERATE_ASSERT_TYPE_CASE(WARN_ ## assert_type); \
        DOCTEST_GENERATE_ASSERT_TYPE_CASE(CHECK_ ## assert_type); \
        DOCTEST_GENERATE_ASSERT_TYPE_CASE(REQUIRE_ ## assert_type)
    switch(at) {
        DOCTEST_GENERATE_ASSERT_TYPE_CASE(WARN);
        DOCTEST_GENERATE_ASSERT_TYPE_CASE(CHECK);
        DOCTEST_GENERATE_ASSERT_TYPE_CASE(REQUIRE);

        DOCTEST_GENERATE_ASSERT_TYPE_CASES(FALSE);

        DOCTEST_GENERATE_ASSERT_TYPE_CASES(THROWS);

        DOCTEST_GENERATE_ASSERT_TYPE_CASES(THROWS_AS);

        DOCTEST_GENERATE_ASSERT_TYPE_CASES(THROWS_WITH);

        DOCTEST_GENERATE_ASSERT_TYPE_CASES(THROWS_WITH_AS);

        DOCTEST_GENERATE_ASSERT_TYPE_CASES(NOTHROW);

        DOCTEST_GENERATE_ASSERT_TYPE_CASES(EQ);
        DOCTEST_GENERATE_ASSERT_TYPE_CASES(NE);
        DOCTEST_GENERATE_ASSERT_TYPE_CASES(GT);
        DOCTEST_GENERATE_ASSERT_TYPE_CASES(LT);
        DOCTEST_GENERATE_ASSERT_TYPE_CASES(GE);
        DOCTEST_GENERATE_ASSERT_TYPE_CASES(LE);

        DOCTEST_GENERATE_ASSERT_TYPE_CASES(UNARY);
        DOCTEST_GENERATE_ASSERT_TYPE_CASES(UNARY_FALSE);

        default: DOCTEST_INTERNAL_ERROR("Tried stringifying invalid assert type!");
    }
    DOCTEST_MSVC_SUPPRESS_WARNING_POP
}
// clang-format on

const char* failureString(assertType::Enum at) {
    if(at & assertType::is_warn)
        return "WARNING";
    if(at & assertType::is_check)
        return "ERROR";
    if(at & assertType::is_require)
        return "FATAL ERROR";
    return "";
}

DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wnull-dereference")
DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH("-Wnull-dereference")
// depending on the current options this will remove the path of filenames
const char* skipPathFromFilename(const char* file) {
#ifndef DOCTEST_CONFIG_DISABLE
    if(getContextOptions()->no_path_in_filenames) {
        auto back    = std::strrchr(file, '\\');
        auto forward = std::strrchr(file, '/');
        if(back || forward) {
            if(back > forward)
                forward = back;
            return forward + 1;
        }
    } else {
        const auto prefixes = getContextOptions()->strip_file_prefixes;
        const char separator = DOCTEST_CONFIG_OPTIONS_FILE_PREFIX_SEPARATOR;
        String::size_type longest_match = 0U;
        for(String::size_type pos = 0U; pos < prefixes.size(); ++pos)
        {
            const auto prefix_start = pos;
            pos = std::min(prefixes.find(separator, prefix_start), prefixes.size());

            const auto prefix_size = pos - prefix_start;
            if(prefix_size > longest_match)
            {
                // TODO under DOCTEST_MSVC: does the comparison need strnicmp() to work with drive
      letter capitalization?
                if(0 == std::strncmp(prefixes.c_str() + prefix_start, file, prefix_size))
                {
                    longest_match = prefix_size;
                }
            }
        }
        return &file[longest_match];
    }
#endif // DOCTEST_CONFIG_DISABLE
    return file;
}
DOCTEST_CLANG_SUPPRESS_WARNING_POP
DOCTEST_GCC_SUPPRESS_WARNING_POP

bool SubcaseSignature::operator==(const SubcaseSignature& other) const {
    return m_line == other.m_line
        && std::strcmp(m_file, other.m_file) == 0
        && m_name == other.m_name;
}

bool SubcaseSignature::operator<(const SubcaseSignature& other) const {
    if(m_line != other.m_line)
        return m_line < other.m_line;
    if(std::strcmp(m_file, other.m_file) != 0)
        return std::strcmp(m_file, other.m_file) < 0;
    return m_name.compare(other.m_name) < 0;
}

DOCTEST_DEFINE_INTERFACE(IContextScope)

namespace detail {
    void filldata<const void*>::fill(std::ostream* stream, const void* in) {
        if (in) { *stream « in; }
        else { *stream « "nullptr"; }
    }

    template <typename T>
    String toStreamLit(T t) {
        std::ostream* os = tlssPush();
        os->operator«(t);
        return tlssPop();
    }
}

#ifdef DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
String toString(const char* in) { return String("\"") + (in ? in : "{null string}") + "\""; }
#endif // DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING

#if DOCTEST_MSVC >= DOCTEST_COMPILER(19, 20, 0)
// see this issue on why this is needed: https://github.com/doctest/doctest/issues/183
String toString(const std::string& in) { return in.c_str(); }
#endif // VS 2019

String toString(String in) { return in; }

String toString(std::nullptr_t) { return "nullptr"; }

String toString(bool in) { return in ? "true" : "false"; }

String toString(float in) { return toStreamLit(in); }
String toString(double in) { return toStreamLit(in); }
String toString(double long in) { return toStreamLit(in); }

String toString(char in) { return toStreamLit(static_cast<signed>(in)); }
String toString(char signed in) { return toStreamLit(static_cast<signed>(in)); }
String toString(char unsigned in) { return toStreamLit(static_cast<unsigned>(in)); }
String toString(short in) { return toStreamLit(in); }
String toString(short unsigned in) { return toStreamLit(in); }
String toString(signed in) { return toStreamLit(in); }
String toString(unsigned in) { return toStreamLit(in); }
String toString(long in) { return toStreamLit(in); }
String toString(long unsigned in) { return toStreamLit(in); }
String toString(long long in) { return toStreamLit(in); }
String toString(long long unsigned in) { return toStreamLit(in); }

Approx::Approx(double value)
        : m_epsilon(static_cast<double>(std::numeric_limits<float>::epsilon()) * 100)
        , m_scale(1.0)
        , m_value(value) {}

Approx Approx::operator()(double value) const {
    Approx approx(value);
    approx.epsilon(m_epsilon);
    approx.scale(m_scale);
    return approx;
}

Approx& Approx::epsilon(double newEpsilon) {
    m_epsilon = newEpsilon;
    return *this;
}
Approx& Approx::scale(double newScale) {
    m_scale = newScale;
    return *this;
}

bool operator==(double lhs, const Approx& rhs) {
    // Thanks to Richard Harris for his help refining this formula
    return std::fabs(lhs - rhs.m_value) <
           rhs.m_epsilon * (rhs.m_scale + std::max<double>(std::fabs(lhs), std::fabs(rhs.m_value)));
}
bool operator==(const Approx& lhs, double rhs) { return operator==(rhs, lhs); }
bool operator!=(double lhs, const Approx& rhs) { return !operator==(lhs, rhs); }
bool operator!=(const Approx& lhs, double rhs) { return !operator==(rhs, lhs); }
bool operator<=(double lhs, const Approx& rhs) { return lhs < rhs.m_value || lhs == rhs; }
bool operator<=(const Approx& lhs, double rhs) { return lhs.m_value < rhs || lhs == rhs; }
bool operator>=(double lhs, const Approx& rhs) { return lhs > rhs.m_value || lhs == rhs; }
bool operator>=(const Approx& lhs, double rhs) { return lhs.m_value > rhs || lhs == rhs; }
bool operator<(double lhs, const Approx& rhs) { return lhs < rhs.m_value && lhs != rhs; }
bool operator<(const Approx& lhs, double rhs) { return lhs.m_value < rhs && lhs != rhs; }
bool operator>(double lhs, const Approx& rhs) { return lhs > rhs.m_value && lhs != rhs; }
bool operator>(const Approx& lhs, double rhs) { return lhs.m_value > rhs && lhs != rhs; }

String toString(const Approx& in) {
    return "Approx( " + doctest::toString(in.m_value) + " )";
}
const ContextOptions* getContextOptions() { return DOCTEST_BRANCH_ON_DISABLED(nullptr, g_cs); }

DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4738)
template <typename F>
IsNaN<F>::operator bool() const {
    return std::isnan(value) ^ flipped;
}
DOCTEST_MSVC_SUPPRESS_WARNING_POP
template struct DOCTEST_INTERFACE_DEF IsNaN<float>;
template struct DOCTEST_INTERFACE_DEF IsNaN<double>;
template struct DOCTEST_INTERFACE_DEF IsNaN<long double>;
template <typename F>
String toString(IsNaN<F> in) { return String(in.flipped ? "! " : "") + "IsNaN( " +
doctest::toString(in.value) + " )"; }
String toString(IsNaN<float> in) { return toString<float>(in); }
String toString(IsNaN<double> in) { return toString<double>(in); }
String toString(IsNaN<double long> in) { return toString<double long>(in); }

} // namespace doctest

#ifdef DOCTEST_CONFIG_DISABLE
namespace doctest {
Context::Context(int, const char* const*) {}
Context::~Context() = default;
void Context::applyCommandLine(int, const char* const*) {}
void Context::addFilter(const char*, const char*) {}
void Context::clearFilters() {}
void Context::setOption(const char*, bool) {}
void Context::setOption(const char*, int) {}
void Context::setOption(const char*, const char*) {}
bool Context::shouldExit() { return false; }
void Context::setAsDefaultForAssertsOutOfTestCases() {}
void Context::setAssertHandler(detail::assert_handler) {}
void Context::setCout(std::ostream*) {}
int  Context::run() { return 0; }

int                         IReporter::get_num_active_contexts() { return 0; }
const IContextScope* const* IReporter::get_active_contexts() { return nullptr; }
int                         IReporter::get_num_stringified_contexts() { return 0; }
const String*               IReporter::get_stringified_contexts() { return nullptr; }
```

```

int registerReporter(const char*, int, IReporter*) { return 0; }

} // namespace doctest
#else // DOCTEST_CONFIG_DISABLE

#if !defined(DOCTEST_CONFIG_COLORS_NONE)
#if !defined(DOCTEST_CONFIG_COLORS_WINDOWS) && !defined(DOCTEST_CONFIG_COLORS_ANSI)
#ifdef DOCTEST_PLATFORM_WINDOWS
#define DOCTEST_CONFIG_COLORS_WINDOWS
#else // linux
#define DOCTEST_CONFIG_COLORS_ANSI
#endif // platform
#endif // DOCTEST_CONFIG_COLORS_WINDOWS && DOCTEST_CONFIG_COLORS_ANSI
#endif // DOCTEST_CONFIG_COLORS_NONE

namespace doctest_detail_test_suite_ns {
// holds the current test suite
doctest::detail::TestSuite& getCurrentTestSuite() {
    static doctest::detail::TestSuite data{};
    return data;
}
} // namespace doctest_detail_test_suite_ns

namespace doctest {
namespace {
    // the int (priority) is part of the key for automatic sorting - sadly one can register a
    // reporter with a duplicate name and a different priority but hopefully that won't happen often
:|
    using reporterMap = std::map<std::pair<int, String>, reporterCreatorFunc>;

    reporterMap& getReporters() {
        static reporterMap data;
        return data;
    }
    reporterMap& getListeners() {
        static reporterMap data;
        return data;
    }
} // namespace
namespace detail {
#define DOCTEST_ITERATE_THROUGH_REPORTERS(function, ...)                                      \
    for(auto& curr_rep : g_cs->reporters_currently_used)                                    \
    curr_rep->function(__VA_ARGS__)

    bool checkIfShouldThrow(assertType::Enum at) {
        if(at & assertType::is_require)
            return true;

        if((at & assertType::is_check)
           && getContextOptions()->abort_after > 0 &&
           (g_cs->numAssertsFailed + g_cs->numAssertsFailedCurrentTest_atomic) >=
                   getContextOptions()->abort_after)
            return true;

        return false;
    }

#ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
    DOCTEST_NORETURN void throwException() {
        g_cs->shouldLogCurrentException = false;
        throw TestFailureException(); // NOLINT(hicpp-exception-baseclass)
    }
#else // DOCTEST_CONFIG_NO_EXCEPTIONS
    void throwException() {}
#endif // DOCTEST_CONFIG_NO_EXCEPTIONS
} // namespace detail

namespace {
    using namespace detail;
    // matching of a string against a wildcard mask (case sensitivity configurable) taken from
    // https://www.codeproject.com/Articles/1088/Wildcard-string-compare-globbing
    int wildcmp(const char* str, const char* wild, bool caseSensitive) {
        const char* cp = str;
        const char* mp = wild;

        while((*str) && (*wild != '*')) {
            if((caseSensitive ? (*wild != *str) : (tolower(*wild) != tolower(*str))) &&
               (*wild != '?')) {
                return 0;
            }
            wild++;
            str++;
        }

        while(*str) {
            if(*wild == '*') {
                if(!*++wild) {
                    return 1;
                }
                mp = wild;
                cp = str + 1;
            } else if((caseSensitive ? (*wild == *str) : (tolower(*wild) == tolower(*str))) ||
                      (*wild == '?')) {
                wild++;
                str++;
            } else {
                wild = mp;
                str  = cp++;
            }
        }

        while(*wild == '*') {
            wild++;
        }
        return !*wild;
    }

    // checks if the name matches any of the filters (and can be configured what to do when empty)
    bool matchesAny(const char* name, const std::vector<String>& filters, bool matchEmpty,
        bool caseSensitive) {
        if (filters.empty() && matchEmpty)
            return true;
        for (auto& curr : filters)
            if (wildcmp(name, curr.c_str(), caseSensitive))
                return true;
        return false;
    }

    DOCTEST_NO_SANITIZE_INTEGER
    unsigned long long hash(unsigned long long a, unsigned long long b) {
        return (a « 5) + b;
    }

    // C string hash function (djb2) - taken from http://www.cse.yorku.ca/~oz/hash.html
    DOCTEST_NO_SANITIZE_INTEGER
    unsigned long long hash(const char* str) {
        unsigned long long hash = 5381;
        char c;
        while ((c = *str++))
            hash = ((hash « 5) + hash) + c; // hash * 33 + c
        return hash;
    }

    unsigned long long hash(const SubcaseSignature& sig) {
        return hash(hash(hash(sig.m_file), hash(sig.m_name.c_str())), sig.m_line);
    }

    unsigned long long hash(const std::vector<SubcaseSignature>& sigs, size_t count) {
        unsigned long long running = 0;
        auto end = sigs.begin() + count;
        for (auto it = sigs.begin(); it != end; it++) {
            running = hash(running, hash(*it));
        }
        return running;
    }

    unsigned long long hash(const std::vector<SubcaseSignature>& sigs) {
        unsigned long long running = 0;
        for (const SubcaseSignature& sig : sigs) {
            running = hash(running, hash(sig));
        }
        return running;
    }
} // namespace
namespace detail {
    bool Subcase::checkFilters() {
        if (g_cs->subcaseStack.size() < size_t(g_cs->subcase_filter_levels)) {
            if (!matchesAny(m_signature.m_name.c_str(), g_cs->filters[6], true, g_cs->case_sensitive))
                return true;
            if (matchesAny(m_signature.m_name.c_str(), g_cs->filters[7], false, g_cs->case_sensitive))
                return true;
        }
        return false;
    }

    Subcase::Subcase(const String& name, const char* file, int line)
            : m_signature({name, file, line}) {
        if (!g_cs->reachedLeaf) {
            if (g_cs->nextSubcaseStack.size() <= g_cs->subcaseStack.size()
                || g_cs->nextSubcaseStack[g_cs->subcaseStack.size()] == m_signature) {
                // Going down.
                if (checkFilters()) { return; }

                g_cs->subcaseStack.push_back(m_signature);
                g_cs->currentSubcaseDepth++;
                m_entered = true;
                DOCTEST_ITERATE_THROUGH_REPORTERS(subcase_start, m_signature);
            }
        } else {
            if (g_cs->subcaseStack[g_cs->currentSubcaseDepth] == m_signature) {
                // This subcase is reentered via control flow.
                g_cs->currentSubcaseDepth++;
                m_entered = true;
                DOCTEST_ITERATE_THROUGH_REPORTERS(subcase_start, m_signature);
            } else if (g_cs->nextSubcaseStack.size() <= g_cs->currentSubcaseDepth
                    && g_cs->fullyTraversedSubcases.find(hash(hash(g_cs->subcaseStack, g_cs->currentSubcaseDepth), hash(m_signature)))
                    == g_cs->fullyTraversedSubcases.end()) {
                if (checkFilters()) { return; }
                // This subcase is part of the one to be executed next.
                g_cs->nextSubcaseStack.clear();
                g_cs->nextSubcaseStack.insert(g_cs->nextSubcaseStack.end(),
                    g_cs->subcaseStack.begin(), g_cs->subcaseStack.begin() + g_cs->currentSubcaseDepth);
                g_cs->nextSubcaseStack.push_back(m_signature);
            }
        }
    }

    DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4996) // std::uncaught_exception is deprecated in C++17
    DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH("-Wdeprecated-declarations")
    DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wdeprecated-declarations")

    Subcase::~Subcase() {
        if (m_entered) {
            g_cs->currentSubcaseDepth--;

            if (!g_cs->reachedLeaf) {
                // Leaf.
                g_cs->fullyTraversedSubcases.insert(hash(g_cs->subcaseStack));
                g_cs->nextSubcaseStack.clear();
                g_cs->reachedLeaf = true;
            } else if (g_cs->nextSubcaseStack.empty()) {
                // All children are finished.
                g_cs->fullyTraversedSubcases.insert(hash(g_cs->subcaseStack));
            }

#if defined(__cpp_lib_uncaught_exceptions) && __cpp_lib_uncaught_exceptions >= 201411L && (!defined(__MAC_OS_X_VERSION_MIN_REQUIRED) || __MAC_OS_X_VERSION_MIN_REQUIRED >= 101200)
            if(std::uncaught_exceptions() > 0
#else
            if(std::uncaught_exception()
#endif
                && g_cs->shouldLogCurrentException) {
                DOCTEST_ITERATE_THROUGH_REPORTERS(
                        test_case_exception, {"exception thrown in subcase - will translate later "
                                                "when the whole test case has been exited (cannot "
                                                "translate while there is an active exception)",
                                                false});
                g_cs->shouldLogCurrentException = false;
            }

            DOCTEST_ITERATE_THROUGH_REPORTERS(subcase_end, DOCTEST_EMPTY);
        }
    }

    DOCTEST_CLANG_SUPPRESS_WARNING_POP
    DOCTEST_GCC_SUPPRESS_WARNING_POP
    DOCTEST_MSVC_SUPPRESS_WARNING_POP

    Subcase::operator bool() const { return m_entered; }

    Result::Result(bool passed, const String& decomposition)
            : m_passed(passed)
            , m_decomp(decomposition) {}

    ExpressionDecomposer::ExpressionDecomposer(assertType::Enum at)
            : m_at(at) {}

    TestSuite& TestSuite::operator*(const char* in) {
        m_test_suite = in;
        return *this;
    }

    TestCase::TestCase(funcType test, const char* file, unsigned line, const TestSuite& test_suite,
                       const String& type, int template_id) {
        m_file              = file;
        m_line              = line;
        m_name              = nullptr; // will be later overridden in operator*
        m_test_suite        = test_suite.m_test_suite;
        m_description       = test_suite.m_description;
        m_skip              = test_suite.m_skip;
        m_no_breaks         = test_suite.m_no_breaks;
        m_no_output         = test_suite.m_no_output;
        m_may_fail          = test_suite.m_may_fail;
        m_should_fail       = test_suite.m_should_fail;
        m_expected_failures = test_suite.m_expected_failures;
        m_timeout           = test_suite.m_timeout;

        m_test        = test;
        m_type        = type;
        m_template_id = template_id;
    }

    TestCase::TestCase(const TestCase& other)
            : TestCaseData() {
        *this = other;
    }

    DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(26434) // hides a non-virtual function
    TestCase& TestCase::operator=(const TestCase& other) {
        TestCaseData::operator=(other);
        m_test        = other.m_test;
        m_type        = other.m_type;
        m_template_id = other.m_template_id;
        m_full_name   = other.m_full_name;

        if(m_template_id != -1)
            m_name = m_full_name.c_str();
        return *this;
    }
    DOCTEST_MSVC_SUPPRESS_WARNING_POP

    TestCase& TestCase::operator*(const char* in) {
        m_name = in;
        // make a new name with an appended type for templated test case
        if(m_template_id != -1) {
            m_full_name = String(m_name) + "<" + m_type + ">";
            // redirect the name to point to the newly constructed full name
            m_name = m_full_name.c_str();
        }
        return *this;
    }

    bool TestCase::operator<(const TestCase& other) const {
        // this will be used only to differentiate between test cases - not relevant for sorting
        if(m_line != other.m_line)
            return m_line < other.m_line;
        const int name_cmp = strcmp(m_name, other.m_name);
        if(name_cmp != 0)
            return name_cmp < 0;
        const int file_cmp = m_file.compare(other.m_file);
        if(file_cmp != 0)
            return file_cmp < 0;
        return m_template_id < other.m_template_id;
    }

    // all the registered tests
    std::set<TestCase>& getRegisteredTests() {
        static std::set<TestCase> data;
        return data;
    }
} // namespace detail
namespace {
    using namespace detail;
    // for sorting tests by file/line
    bool fileOrderComparator(const TestCase* lhs, const TestCase* rhs) {
        // this is needed because MSVC gives different case for drive letters
        // for __FILE__ when evaluated in a header and a source file
        const int res = lhs->m_file.compare(rhs->m_file, bool(DOCTEST_MSVC));
        if(res != 0)
            return res < 0;
        if(lhs->m_line != rhs->m_line)
            return lhs->m_line < rhs->m_line;
        return lhs->m_template_id < rhs->m_template_id;
    }

    // for sorting tests by suite/file/line
    bool suiteOrderComparator(const TestCase* lhs, const TestCase* rhs) {
        const int res = std::strcmp(lhs->m_test_suite, rhs->m_test_suite);
        if(res != 0)
            return res < 0;
        return fileOrderComparator(lhs, rhs);
    }

    // for sorting tests by name/suite/file/line
    bool nameOrderComparator(const TestCase* lhs, const TestCase* rhs) {
        const int res = std::strcmp(lhs->m_name, rhs->m_name);
        if(res != 0)
            return res < 0;
        return suiteOrderComparator(lhs, rhs);
    }

    DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wdeprecated-declarations")
    void color_to_stream(std::ostream& s, Color::Enum code) {
        static_cast<void>(s);    // for DOCTEST_CONFIG_COLORS_NONE or DOCTEST_CONFIG_COLORS_WINDOWS
        static_cast<void>(code); // for DOCTEST_CONFIG_COLORS_NONE
#ifdef DOCTEST_CONFIG_COLORS_ANSI
        if(g_no_colors ||
           (isatty(STDOUT_FILENO) == false && getContextOptions()->force_colors == false))
            return;

        auto col = "";
        // clang-format off
            switch(code) {
                case Color::Red:         col = "[0;31m"; break;
                case Color::Green:       col = "[0;32m"; break;
                case Color::Blue:        col = "[0;34m"; break;
                case Color::Cyan:        col = "[0;36m"; break;
                case Color::Yellow:      col = "[0;33m"; break;
                case Color::Grey:        col = "[1;30m"; break;
                case Color::LightGrey:   col = "[0;37m"; break;
                case Color::BrightRed:   col = "[1;31m"; break;
                case Color::BrightGreen: col = "[1;32m"; break;
                case Color::BrightWhite: col = "[1;37m"; break;
                case Color::Bright: // invalid
                case Color::None:
                case Color::White:
                default:                 col = "[0m";
            }
        // clang-format on
        s « "\033" « col;
#endif // DOCTEST_CONFIG_COLORS_ANSI

#ifdef DOCTEST_CONFIG_COLORS_WINDOWS
        if(g_no_colors ||
           (_isatty(_fileno(stdout)) == false && getContextOptions()->force_colors == false))
            return;

        static struct ConsoleHelper {
            HANDLE stdoutHandle;
            WORD   origFgAttrs;
            WORD   origBgAttrs;

            ConsoleHelper() {
                stdoutHandle = GetStdHandle(STD_OUTPUT_HANDLE);
                CONSOLE_SCREEN_BUFFER_INFO csbiInfo;
                GetConsoleScreenBufferInfo(stdoutHandle, &csbiInfo);
                origFgAttrs = csbiInfo.wAttributes & ~(BACKGROUND_GREEN | BACKGROUND_RED |
                    BACKGROUND_BLUE | BACKGROUND_INTENSITY);
                origBgAttrs = csbiInfo.wAttributes & ~(FOREGROUND_GREEN | FOREGROUND_RED |
                    FOREGROUND_BLUE | FOREGROUND_INTENSITY);
            }
        } ch;

#define DOCTEST_SET_ATTR(x) SetConsoleTextAttribute(ch.stdoutHandle, x | ch.origBgAttrs)

        // clang-format off
        switch (code) {
            case Color::White:       DOCTEST_SET_ATTR(FOREGROUND_GREEN | FOREGROUND_RED | FOREGROUND_BLUE); break;
            case Color::Red:         DOCTEST_SET_ATTR(FOREGROUND_RED); break;
            case Color::Green:       DOCTEST_SET_ATTR(FOREGROUND_GREEN); break;
            case Color::Blue:        DOCTEST_SET_ATTR(FOREGROUND_BLUE); break;
            case Color::Cyan:        DOCTEST_SET_ATTR(FOREGROUND_BLUE | FOREGROUND_GREEN); break;
            case Color::Yellow:      DOCTEST_SET_ATTR(FOREGROUND_RED | FOREGROUND_GREEN); break;
            case Color::Grey:        DOCTEST_SET_ATTR(0); break;
            case Color::LightGrey:   DOCTEST_SET_ATTR(FOREGROUND_INTENSITY); break;
            case Color::BrightRed:   DOCTEST_SET_ATTR(FOREGROUND_INTENSITY | FOREGROUND_RED); break;
            case Color::BrightGreen: DOCTEST_SET_ATTR(FOREGROUND_INTENSITY | FOREGROUND_GREEN); break;
            case Color::BrightWhite: DOCTEST_SET_ATTR(FOREGROUND_INTENSITY | FOREGROUND_GREEN | FOREGROUND_RED | FOREGROUND_BLUE); break;
            case Color::None:
            case Color::Bright: // invalid
            default:                 DOCTEST_SET_ATTR(ch.origFgAttrs);
        }
            // clang-format on
#endif // DOCTEST_CONFIG_COLORS_WINDOWS
    }
    DOCTEST_CLANG_SUPPRESS_WARNING_POP

    std::vector<const IExceptionTranslator*>& getExceptionTranslators() {
        static std::vector<const IExceptionTranslator*> data;
        return data;
    }

    String translateActiveException() {
#ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
        String res;
        auto&  translators = getExceptionTranslators();
        for(auto& curr : translators)
            if(curr->translate(res))
                return res;
        // clang-format off
        DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH("-Wcatch-value")
        try {
            throw;
        } catch(std::exception& ex) {
            return ex.what();
        } catch(std::string& msg) {
            return msg.c_str();
        } catch(const char* msg) {
            return msg;
        } catch(...) {
            return "unknown exception";
        }
        DOCTEST_GCC_SUPPRESS_WARNING_POP
// clang-format on
#else  // DOCTEST_CONFIG_NO_EXCEPTIONS
        return "";
#endif // DOCTEST_CONFIG_NO_EXCEPTIONS
    }
} // namespace

namespace detail {
    // used by the macros for registering tests
    int regTest(const TestCase& tc) {
        getRegisteredTests().insert(tc);
        return 0;
    }

    // sets the current test suite
    int setTestSuite(const TestSuite& ts) {
        doctest_detail_test_suite_ns::getCurrentTestSuite() = ts;
        return 0;
    }

#ifdef DOCTEST_IS_DEBUGGER_ACTIVE
    bool isDebuggerActive() { return DOCTEST_IS_DEBUGGER_ACTIVE(); }
#else // DOCTEST_IS_DEBUGGER_ACTIVE
#ifdef DOCTEST_PLATFORM_LINUX
    class ErrnoGuard {
    public:
        ErrnoGuard() : m_oldErrno(errno) {}
        ~ErrnoGuard() { errno = m_oldErrno; }
    private:
        int m_oldErrno;
    };
    // See the comments in Catch2 for the reasoning behind this implementation:
    // https://github.com/catchorg/Catch2/blob/v2.13.1/include/internal/catch_debugger.cpp#L79-L102
    bool isDebuggerActive() {
        ErrnoGuard guard;
        std::ifstream in("/proc/self/status");
        for(std::string line; std::getline(in, line);) {
            static const int PREFIX_LEN = 11;
            if(line.compare(0, PREFIX_LEN, "TracerPid:\t") == 0) {
                return line.length() > PREFIX_LEN && line[PREFIX_LEN] != '0';
            }
        }
        return false;
    }
#elif defined(DOCTEST_PLATFORM_MAC)
    // The following function is taken directly from the following technical note:
    // https://developer.apple.com/library/archive/qa/qa1361/_index.html
    // Returns true if the current process is being debugged (either
    // running under the debugger or has a debugger attached post facto).
    bool isDebuggerActive() {
        int        mib[4];
        kinfo_proc info;
        size_t     size;
        // Initialize the flags so that, if sysctl fails for some bizarre
        // reason, we get a predictable result.
        info.kp_proc.p_flag = 0;
        // Initialize mib, which tells sysctl the info we want, in this case
        // we're looking for information about a specific process ID.
        mib[0] = CTL_KERN;
        mib[1] = KERN_PROC;
        mib[2] = KERN_PROC_PID;
        mib[3] = getpid();
        // Call sysctl.
        size = sizeof(info);
        if(sysctl(mib, DOCTEST_COUNTOF(mib), &info, &size, 0, 0) != 0) {
            std::cerr « "\nCall to sysctl failed - unable to determine if debugger is active **\n";
            return false;
        }
        // We're being debugged if the P_TRACED flag is set.
        return ((info.kp_proc.p_flag & P_TRACED) != 0);
    }
#elif DOCTEST_MSVC || defined(__MINGW32__) || defined(__MINGW64__)
    bool isDebuggerActive() { return ::IsDebuggerPresent() != 0; }
#else
    bool isDebuggerActive() { return false; }
#endif // Platform
#endif // DOCTEST_IS_DEBUGGER_ACTIVE

    void registerExceptionTranslatorImpl(const IExceptionTranslator* et) {
        if(std::find(getExceptionTranslators().begin(), getExceptionTranslators().end(), et) ==
           getExceptionTranslators().end())
            getExceptionTranslators().push_back(et);
    }

    DOCTEST_THREAD_LOCAL std::vector<IContextScope*> g_infoContexts; // for logging with INFO()

    ContextScopeBase::ContextScopeBase() {
        g_infoContexts.push_back(this);
    }

    ContextScopeBase::ContextScopeBase(ContextScopeBase&& other) noexcept {
        if (other.need_to_destroy) {
            other.destroy();
        }
        other.need_to_destroy = false;
        g_infoContexts.push_back(this);
    }

    DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4996) // std::uncaught_exception is deprecated in C++17
    DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH("-Wdeprecated-declarations")
    DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wdeprecated-declarations")

    // destroy cannot be inlined into the destructor because that would mean calling stringify after
    // ContextScope has been destroyed (base class destructors run after derived class destructors).
    // Instead, ContextScope calls this method directly from its destructor.
    void ContextScopeBase::destroy() {
#if defined(__cpp_lib_uncaught_exceptions) && __cpp_lib_uncaught_exceptions >= 201411L &&
(!defined(__MAC_OS_X_VERSION_MIN_REQUIRED) || __MAC_OS_X_VERSION_MIN_REQUIRED >= 101200)
        if(std::uncaught_exceptions() > 0) {
#else
        if(std::uncaught_exception()) {
#endif
            std::ostringstream s;
            this->stringify(&s);
            g_cs->stringifiedContexts.push_back(s.str().c_str());
        }
        g_infoContexts.pop_back();
    }

    DOCTEST_CLANG_SUPPRESS_WARNING_POP
    DOCTEST_GCC_SUPPRESS_WARNING_POP
    DOCTEST_MSVC_SUPPRESS_WARNING_POP
} // namespace detail
namespace {
    using namespace detail;

#if !defined(DOCTEST_CONFIG_POSIX_SIGNALS) && !defined(DOCTEST_CONFIG_WINDOWS_SEH)
    struct FatalConditionHandler
    {
        static void reset() {}
        static void allocateAltStackMem() {}
        static void freeAltStackMem() {}
    };
#else // DOCTEST_CONFIG_POSIX_SIGNALS || DOCTEST_CONFIG_WINDOWS_SEH

    void reportFatal(const std::string&);

#ifdef DOCTEST_PLATFORM_WINDOWS

    struct SignalDefs
    {
        DWORD id;
        const char* name;
    };
    // There is no 1-1 mapping between signals and windows exceptions.
    // Windows can easily distinguish between SO and SigSegV,
    // but SigInt, SigTerm, etc are handled differently.
    SignalDefs signalDefs[] = {
            {static_cast<DWORD>(EXCEPTION_ILLEGAL_INSTRUCTION),
             "SIGILL - Illegal instruction signal"},
            {static_cast<DWORD>(EXCEPTION_STACK_OVERFLOW), "SIGSEGV - Stack overflow"},
            {static_cast<DWORD>(EXCEPTION_ACCESS_VIOLATION),
             "SIGSEGV - Segmentation violation signal"},
            {static_cast<DWORD>(EXCEPTION_INT_DIVIDE_BY_ZERO), "Divide by zero error"},
    };

    struct FatalConditionHandler
    {
        static LONG CALLBACK handleException(PEXCEPTION_POINTERS ExceptionInfo) {
            // Multiple threads may enter this filter/handler at once. We want the error message to be
printed on the
            // console just once no matter how many threads have crashed.
            DOCTEST_DECLARE_STATIC_MUTEX(mutex)
            static bool execute = true;
            {
                DOCTEST_LOCK_MUTEX(mutex)
                if(execute) {
                    bool reported = false;
                    for(size_t i = 0; i < DOCTEST_COUNTOF(signalDefs); ++i) {
                        if(ExceptionInfo->ExceptionRecord->ExceptionCode == signalDefs[i].id) {
                            reportFatal(signalDefs[i].name);
                            reported = true;
                            break;
                        }
                    }
                    if(reported == false)
                        reportFatal("Unhandled SEH exception caught");
                    if(isDebuggerActive() && !g_cs->no_breaks)
                        DOCTEST_BREAK_INTO_DEBUGGER();
                }
                execute = false;
            }
            std::exit(EXIT_FAILURE);
        }

        static void allocateAltStackMem() {}
        static void freeAltStackMem() {}

        FatalConditionHandler() {
            isSet = true;
            // 32k seems enough for doctest to handle stack overflow,
            // but the value was found experimentally, so there is no strong guarantee
            guaranteeSize = 32 * 1024;
            // Register an unhandled exception filter
            previousTop = SetUnhandledExceptionFilter(handleException);
            // Pass in guarantee size to be filled
            SetThreadStackGuarantee(&guaranteeSize);

            // On Windows uncaught exceptions from another thread, exceptions from
            // destructors, or calls to std::terminate are not a SEH exception

            // The terminal handler gets called when:
            // - std::terminate is called FROM THE TEST RUNNER THREAD
            // - an exception is thrown from a destructor FROM THE TEST RUNNER THREAD
            original_terminate_handler = std::get_terminate();
            std::set_terminate([]() DOCTEST_NOEXCEPT {
                reportFatal("Terminate handler called");
                if(isDebuggerActive() && !g_cs->no_breaks)
                    DOCTEST_BREAK_INTO_DEBUGGER();
                std::exit(EXIT_FAILURE); // explicitly exit - otherwise the SIGABRT handler may be
called as well
            });

            // SIGABRT is raised when:
            // - std::terminate is called FROM A DIFFERENT THREAD
            // - an exception is thrown from a destructor FROM A DIFFERENT THREAD
            // - an uncaught exception is thrown FROM A DIFFERENT THREAD
            prev_sigabrt_handler = std::signal(SIGABRT, [](int signal) DOCTEST_NOEXCEPT {
                if(signal == SIGABRT) {
                    reportFatal("SIGABRT - Abort (abnormal termination) signal");
                    if(isDebuggerActive() && !g_cs->no_breaks)
                        DOCTEST_BREAK_INTO_DEBUGGER();
                    std::exit(EXIT_FAILURE);
                }
            });

            // The following settings are taken from google test, and more
            // specifically from UnitTest::Run() inside of gtest.cc

            // the user does not want to see pop-up dialogs about crashes
            prev_error_mode_1 = SetErrorMode(SEM_FAILCRITICALERRORS | SEM_NOALIGNMENTFAULTEXCEPT |
                                             SEM_NOGPFAULTERRORBOX | SEM_NOOPENFILEERRORBOX);
            // This forces the abort message to go to stderr in all circumstances.
            prev_error_mode_2 = _set_error_mode(_OUT_TO_STDERR);
            // In the debug version, Visual Studio pops up a separate dialog
            // offering a choice to debug the aborted program - we want to disable that.
            prev_abort_behavior = _set_abort_behavior(0x0, _WRITE_ABORT_MSG | _CALL_REPORTFAULT);
            // In debug mode, the Windows CRT can crash with an assertion over invalid
            // input (e.g. passing an invalid file descriptor). The default handling
            // for these assertions is to pop up a dialog and wait for user input.
            // Instead ask the CRT to dump such assertions to stderr non-interactively.
            prev_report_mode = _CrtSetReportMode(_CRT_ASSERT, _CRTDBG_MODE_FILE | _CRTDBG_MODE_DEBUG);
            prev_report_file = _CrtSetReportFile(_CRT_ASSERT, _CRTDBG_FILE_STDERR);
        }

        static void reset() {
            if(isSet) {
                // Unregister handler and restore the old guarantee
                SetUnhandledExceptionFilter(previousTop);
                SetThreadStackGuarantee(&guaranteeSize);
                std::set_terminate(original_terminate_handler);
                std::signal(SIGABRT, prev_sigabrt_handler);
                SetErrorMode(prev_error_mode_1);
                _set_error_mode(prev_error_mode_2);
                _set_abort_behavior(prev_abort_behavior, _WRITE_ABORT_MSG | _CALL_REPORTFAULT);
                static_cast<void>(_CrtSetReportMode(_CRT_ASSERT, prev_report_mode));
                static_cast<void>(_CrtSetReportFile(_CRT_ASSERT, prev_report_file));
                isSet = false;
            }
        }

        ~FatalConditionHandler() { reset(); }

    private:
        static UINT         prev_error_mode_1;
        static int          prev_error_mode_2;
        static unsigned int prev_abort_behavior;
        static int          prev_report_mode;
        static _HFILE       prev_report_file;
        static void (DOCTEST_CDECL *prev_sigabrt_handler)(int);
        static std::terminate_handler original_terminate_handler;
        static bool isSet;
        static ULONG guaranteeSize;
        static LPTOP_LEVEL_EXCEPTION_FILTER previousTop;
    };

    UINT         FatalConditionHandler::prev_error_mode_1;
    int          FatalConditionHandler::prev_error_mode_2;
    unsigned int FatalConditionHandler::prev_abort_behavior;
    int          FatalConditionHandler::prev_report_mode;
    _HFILE       FatalConditionHandler::prev_report_file;
    void (DOCTEST_CDECL *FatalConditionHandler::prev_sigabrt_handler)(int);
    std::terminate_handler FatalConditionHandler::original_terminate_handler;
    bool FatalConditionHandler::isSet = false;
    ULONG FatalConditionHandler::guaranteeSize = 0;
    LPTOP_LEVEL_EXCEPTION_FILTER FatalConditionHandler::previousTop = nullptr;

#else // DOCTEST_PLATFORM_WINDOWS

    struct SignalDefs
    {
        int         id;
        const char* name;
    };
    SignalDefs signalDefs[] = {{SIGINT, "SIGINT - Terminal interrupt signal"},
                               {SIGILL, "SIGILL - Illegal instruction signal"},
                               {SIGFPE, "SIGFPE - Floating point error signal"},
                               {SIGSEGV, "SIGSEGV - Segmentation violation signal"},
                               {SIGTERM, "SIGTERM - Termination request signal"},
                               {SIGABRT, "SIGABRT - Abort (abnormal termination) signal"}};

    struct FatalConditionHandler
    {
        static bool             isSet;
        static struct sigaction oldSigActions[DOCTEST_COUNTOF(signalDefs)];
        static stack_t          oldSigStack;
        static size_t           altStackSize;
        static char*            altStackMem;

        static void handleSignal(int sig) {
            const char* name = "<unknown signal>";
            for(std::size_t i = 0; i < DOCTEST_COUNTOF(signalDefs); ++i) {
                SignalDefs& def = signalDefs[i];
                if(sig == def.id) {
                    name = def.name;
                    break;
                }
            }
            reset();
            reportFatal(name);
            raise(sig);
        }

        static void allocateAltStackMem() {
            altStackMem = new char[altStackSize];
        }

        static void freeAltStackMem() {
            delete[] altStackMem;
        }

        FatalConditionHandler() {
            isSet = true;
            stack_t sigStack;
            sigStack.ss_sp    = altStackMem;
            sigStack.ss_size  = altStackSize;
            sigStack.ss_flags = 0;
            sigaltstack(&sigStack, &oldSigStack);
            struct sigaction sa = {};
            sa.sa_handler       = handleSignal;
            sa.sa_flags         = SA_ONSTACK;
            for(std::size_t i = 0; i < DOCTEST_COUNTOF(signalDefs); ++i) {
                sigaction(signalDefs[i].id, &sa, &oldSigActions[i]);
            }
        }

        ~FatalConditionHandler() { reset(); }
        static void reset() {
            if(isSet) {
                // Set signals back to previous values -- hopefully nobody overwrote them in the
meantime
                for(std::size_t i = 0; i < DOCTEST_COUNTOF(signalDefs); ++i) {
                    sigaction(signalDefs[i].id, &oldSigActions[i], nullptr);
                }
                // Return the old stack
                sigaltstack(&oldSigStack, nullptr);
                isSet = false;
            }
        }
    };

    bool             FatalConditionHandler::isSet = false;
    struct sigaction FatalConditionHandler::oldSigActions[DOCTEST_COUNTOF(signalDefs)] = {};
    stack_t          FatalConditionHandler::oldSigStack = {};
    size_t           FatalConditionHandler::altStackSize = 4 * SIGSTKSZ;
    char*            FatalConditionHandler::altStackMem = nullptr;

#endif // DOCTEST_PLATFORM_WINDOWS
#endif // DOCTEST_CONFIG_POSIX_SIGNALS || DOCTEST_CONFIG_WINDOWS_SEH

} // namespace

namespace {
    using namespace detail;

#ifdef DOCTEST_PLATFORM_WINDOWS
#define DOCTEST_OUTPUT_DEBUG_STRING(text) ::OutputDebugStringA(text)
#else
    // TODO: integration with XCode and other IDEs
#define DOCTEST_OUTPUT_DEBUG_STRING(text)
#endif // Platform

    void addAssert(assertType::Enum at) {
        if((at & assertType::is_warn) == 0)
            g_cs->numAssertsCurrentTest_atomic++;
    }

    void addFailedAssert(assertType::Enum at) {
        if((at & assertType::is_warn) == 0)
            g_cs->numAssertsFailedCurrentTest_atomic++;
    }

#if defined(DOCTEST_CONFIG_POSIX_SIGNALS) || defined(DOCTEST_CONFIG_WINDOWS_SEH)
    void reportFatal(const std::string& message) {
        g_cs->failure_flags |= TestCaseFailureReason::Crash;

        DOCTEST_ITERATE_THROUGH_REPORTERS(test_case_exception, {message.c_str(), true});

        while (g_cs->subcaseStack.size()) {
            g_cs->subcaseStack.pop_back();
            DOCTEST_ITERATE_THROUGH_REPORTERS(subcase_end, DOCTEST_EMPTY);
        }

        g_cs->finalizeTestCaseData();

        DOCTEST_ITERATE_THROUGH_REPORTERS(test_case_end, *g_cs);

        DOCTEST_ITERATE_THROUGH_REPORTERS(test_run_end, *g_cs);
    }
#endif // DOCTEST_CONFIG_POSIX_SIGNALS || DOCTEST_CONFIG_WINDOWS_SEH
} // namespace

AssertData::AssertData(assertType::Enum at, const char* file, int line, const char* expr,
    const char* exception_type, const StringContains& exception_string)
    : m_test_case(g_cs->currentTest), m_at(at), m_file(file), m_line(line), m_expr(expr),
    m_failed(true), m_threw(false), m_threw_as(false), m_exception_type(exception_type),
    m_exception_string(exception_string) {
#if DOCTEST_MSVC
    if (m_expr[0] == ' ') // this happens when variadic macros are disabled under MSVC
        ++m_expr;
#endif // MSVC
}

namespace detail {
    ResultBuilder::ResultBuilder(assertType::Enum at, const char* file, int line, const char* expr,
                                 const char* exception_type, const String& exception_string)
        : AssertData(at, file, line, expr, exception_type, exception_string) { }

    ResultBuilder::ResultBuilder(assertType::Enum at, const char* file, int line, const char* expr,
        const char* exception_type, const Contains& exception_string)
        : AssertData(at, file, line, expr, exception_type, exception_string) { }

    void ResultBuilder::setResult(const Result& res) {
        m_decomp = res.m_decomp;
        m_failed = !res.m_passed;
    }

    void ResultBuilder::translateException() {
        m_threw     = true;
        m_exception = translateActiveException();
    }

    bool ResultBuilder::log() {
        if(m_at & assertType::is_throws) {
            m_failed = !m_threw;
        } else if((m_at & assertType::is_throws_as) && (m_at & assertType::is_throws_with)) {
            m_failed = !m_threw_as || !m_exception_string.check(m_exception);
        } else if(m_at & assertType::is_throws_as) {
            m_failed = !m_threw_as;
        } else if(m_at & assertType::is_throws_with) {
            m_failed = !m_exception_string.check(m_exception);
        } else if(m_at & assertType::is_nothrow) {
            m_failed = m_threw;
        }

        if(m_exception.size())
            m_exception = "\"" + m_exception + "\"";

        if(is_running_in_test) {
            addAssert(m_at);
            DOCTEST_ITERATE_THROUGH_REPORTERS(log_assert, *this);

            if(m_failed)
                addFailedAssert(m_at);
        } else if(m_failed) {
            failed_out_of_a_testing_context(*this);
        }

        return m_failed && isDebuggerActive() && !getContextOptions()->no_breaks &&
            (g_cs->currentTest == nullptr || !g_cs->currentTest->m_no_breaks); // break into debugger
    }

    void ResultBuilder::react() const {
        if(m_failed && checkIfShouldThrow(m_at))
            throwException();
    }

    void failed_out_of_a_testing_context(const AssertData& ad) {
        if(g_cs->ah)
            g_cs->ah(ad);
        else
            std::abort();
    }

    bool decomp_assert(assertType::Enum at, const char* file, int line, const char* expr,
                       const Result& result) {
        bool failed = !result.m_passed;

        // ###################################################################################
        // IF THE DEBUGGER BREAKS HERE - GO 1 LEVEL UP IN THE CALLSTACK FOR THE FAILING ASSERT
        // THIS IS THE EFFECT OF HAVING 'DOCTEST_CONFIG_SUPER_FAST_ASSERTS' DEFINED
        // ###################################################################################
        DOCTEST_ASSERT_OUT_OF_TESTS(result.m_decomp);
        DOCTEST_ASSERT_IN_TESTS(result.m_decomp);
        return !failed;
    }

    MessageBuilder::MessageBuilder(const char* file, int line, assertType::Enum severity) {
        m_stream   = tlssPush();
        m_file     = file;
        m_line     = line;
        m_severity = severity;
    }

    MessageBuilder::~MessageBuilder() {
        if (!logged)
            tlssPop();
    }

    DOCTEST_DEFINE_INTERFACE(IExceptionTranslator)

    bool MessageBuilder::log() {
        if (!logged) {
            m_string = tlssPop();
            logged = true;
        }

        DOCTEST_ITERATE_THROUGH_REPORTERS(log_message, *this);

        const bool isWarn = m_severity & assertType::is_warn;

        // warn is just a message in this context so we don't treat it as an assert
        if(!isWarn) {
            addAssert(m_severity);
            addFailedAssert(m_severity);
        }

        return isDebuggerActive() && !getContextOptions()->no_breaks && !isWarn &&
            (g_cs->currentTest == nullptr || !g_cs->currentTest->m_no_breaks); // break into debugger
    }

    void MessageBuilder::react() {
        if(m_severity & assertType::is_require)
            throwException();
    }
} // namespace detail
namespace {
    using namespace detail;

    // clang-format off

// =================================================================================================
// The following code has been taken verbatim from Catch2/include/internal/catch_xmlwriter.h/cpp
// This is done so cherry-picking bug fixes is trivial - even the style/formatting is untouched.
// =================================================================================================

    class XmlEncode {
    public:
        enum ForWhat { ForTextNodes, ForAttributes };

        XmlEncode( std::string const& str, ForWhat forWhat = ForTextNodes );

        void encodeTo( std::ostream& os ) const;

        friend std::ostream& operator « ( std::ostream& os, XmlEncode const& xmlEncode );

    private:
        std::string m_str;
        ForWhat m_forWhat;
    };

    class XmlWriter {
    public:

        class ScopedElement {
        public:
            ScopedElement( XmlWriter* writer );

            ScopedElement( ScopedElement&& other ) DOCTEST_NOEXCEPT;
            ScopedElement& operator=( ScopedElement&& other ) DOCTEST_NOEXCEPT;

            ~ScopedElement();

            ScopedElement& writeText( std::string const& text, bool indent = true );

            template<typename T>
            ScopedElement& writeAttribute( std::string const& name, T const& attribute ) {
                m_writer->writeAttribute( name, attribute );
                return *this;
            }

        private:
            mutable XmlWriter* m_writer = nullptr;
        };

#ifndef DOCTEST_CONFIG_NO_INCLUDE_IOSTREAM
        XmlWriter( std::ostream& os = std::cout );
#else // DOCTEST_CONFIG_NO_INCLUDE_IOSTREAM
        XmlWriter( std::ostream& os );
#endif // DOCTEST_CONFIG_NO_INCLUDE_IOSTREAM
        ~XmlWriter();

        XmlWriter( XmlWriter const& ) = delete;
        XmlWriter& operator=( XmlWriter const& ) = delete;

        XmlWriter& startElement( std::string const& name );

        ScopedElement scopedElement( std::string const& name );

        XmlWriter& endElement();

        XmlWriter& writeAttribute( std::string const& name, std::string const& attribute );

        XmlWriter& writeAttribute( std::string const& name, const char* attribute );

        XmlWriter& writeAttribute( std::string const& name, bool attribute );

        template<typename T>
        XmlWriter& writeAttribute( std::string const& name, T const& attribute ) {
        std::stringstream rss;
            rss « attribute;
            return writeAttribute( name, rss.str() );
        }

        XmlWriter& writeText( std::string const& text, bool indent = true );

        //XmlWriter& writeComment( std::string const& text );

        //void writeStylesheetRef( std::string const& url );

        //XmlWriter& writeBlankLine();

        void ensureTagClosed();

        void writeDeclaration();

    private:

        void newlineIfNecessary();

        bool m_tagIsOpen = false;
        bool m_needsNewline = false;
        std::vector<std::string> m_tags;
        std::string m_indent;
        std::ostream& m_os;
    };

// =================================================================================================
// The following code has been taken verbatim from Catch2/include/internal/catch_xmlwriter.h/cpp
// This is done so cherry-picking bug fixes is trivial - even the style/formatting is untouched.
// =================================================================================================

using uchar = unsigned char;

namespace {

    size_t trailingBytes(unsigned char c) {
        if ((c & 0xE0) == 0xC0) {
            return 2;
        }
        if ((c & 0xF0) == 0xE0) {
            return 3;
        }
        if ((c & 0xF8) == 0xF0) {
            return 4;
        }
        DOCTEST_INTERNAL_ERROR("Invalid multibyte utf-8 start byte encountered");
    }

    uint32_t headerValue(unsigned char c) {
        if ((c & 0xE0) == 0xC0) {
            return c & 0x1F;
        }
        if ((c & 0xF0) == 0xE0) {
            return c & 0x0F;
        }
        if ((c & 0xF8) == 0xF0) {
            return c & 0x07;
        }
        DOCTEST_INTERNAL_ERROR("Invalid multibyte utf-8 start byte encountered");
    }

    void hexEscapeChar(std::ostream& os, unsigned char c) {
        std::ios_base::fmtflags f(os.flags());
        os « "\\x"
            « std::uppercase « std::hex « std::setfill('0') « std::setw(2)
            « static_cast<int>(c);
        os.flags(f);
    }

} // anonymous namespace

    XmlEncode::XmlEncode( std::string const& str, ForWhat forWhat )
    :   m_str( str ),
        m_forWhat( forWhat )
    {}

    void XmlEncode::encodeTo( std::ostream& os ) const {
        // Apostrophe escaping not necessary if we always use " to write attributes
        // (see: https://www.w3.org/TR/xml/#syntax)

        for( std::size_t idx = 0; idx < m_str.size(); ++ idx ) {
            uchar c = m_str[idx];
            switch (c) {
            case '<':   os « "&lt;"; break;
            case '&':   os « "&amp;"; break;

            case '>':
                // See: https://www.w3.org/TR/xml/#syntax
                if (idx > 2 && m_str[idx - 1] == ']' && m_str[idx - 2] == ']')
                    os « "&gt;";
                else
                    os « c;
                break;

            case '\"':
                if (m_forWhat == ForAttributes)
                    os « "&quot;";
                else
                    os « c;
                break;

            default:
                // Check for control characters and invalid utf-8

                // Escape control characters in standard ascii
                // see
https://stackoverflow.com/questions/404107/why-are-control-characters-illegal-in-xml-1-0
                if (c < 0x09 || (c > 0x0D && c < 0x20) || c == 0x7F) {
                    hexEscapeChar(os, c);
                    break;
                }

                // Plain ASCII: Write it to stream
                if (c < 0x7F) {
                    os « c;
                    break;
                }

                // UTF-8 territory
                // Check if the encoding is valid and if it is not, hex escape bytes.
                // Important: We do not check the exact decoded values for validity, only the encoding
format
                // First check that this bytes is a valid lead byte:
                // This means that it is not encoded as 1111 1XXX
                // Or as 10XX XXXX
                if (c <  0xC0 ||
                    c >= 0xF8) {
                    hexEscapeChar(os, c);
                    break;
                }

                auto encBytes = trailingBytes(c);
                // Are there enough bytes left to avoid accessing out-of-bounds memory?
                if (idx + encBytes - 1 >= m_str.size()) {
                    hexEscapeChar(os, c);
                    break;
                }
                // The header is valid, check data
                // The next encBytes bytes must together be a valid utf-8
                // This means: bitpattern 10XX XXXX and the extracted value is sane (ish)
                bool valid = true;
                uint32_t value = headerValue(c);
                for (std::size_t n = 1; n < encBytes; ++n) {
                    uchar nc = m_str[idx + n];
                    valid &= ((nc & 0xC0) == 0x80);
                    value = (value « 6) | (nc & 0x3F);
                }

                if (
                    // Wrong bit pattern of following bytes
                    (!valid) ||
                    // Overlong encodings
                    (value < 0x80) ||
                    (                 value < 0x800   && encBytes > 2) || // removed "0x80 <= value
&&" because redundant
                    (0x800 < value && value < 0x10000 && encBytes > 3) ||
                    // Encoded value out of range
                    (value >= 0x110000)
                    ) {
                    hexEscapeChar(os, c);
                    break;
                }

                // If we got here, this is in fact a valid(ish) utf-8 sequence
                for (std::size_t n = 0; n < encBytes; ++n) {
                    os « m_str[idx + n];
                }
                idx += encBytes - 1;
                break;
            }
        }
    }

    std::ostream& operator « ( std::ostream& os, XmlEncode const& xmlEncode ) {
        xmlEncode.encodeTo( os );
        return os;
    }

    XmlWriter::ScopedElement::ScopedElement( XmlWriter* writer )
    :   m_writer( writer )
    {}

    XmlWriter::ScopedElement::ScopedElement( ScopedElement&& other ) DOCTEST_NOEXCEPT
    :   m_writer( other.m_writer ){
        other.m_writer = nullptr;
    }
    XmlWriter::ScopedElement& XmlWriter::ScopedElement::operator=( ScopedElement&& other )
DOCTEST_NOEXCEPT {
        if ( m_writer ) {
            m_writer->endElement();
        }
        m_writer = other.m_writer;
        other.m_writer = nullptr;
        return *this;
    }


    XmlWriter::ScopedElement::~ScopedElement() {
        if( m_writer )
            m_writer->endElement();
    }

    XmlWriter::ScopedElement& XmlWriter::ScopedElement::writeText( std::string const& text, bool
indent ) {
        m_writer->writeText( text, indent );
        return *this;
    }

    XmlWriter::XmlWriter( std::ostream& os ) : m_os( os )
    {
        // writeDeclaration(); // called explicitly by the reporters that use the writer class - see
issue #627
    }

    XmlWriter::~XmlWriter() {
        while( !m_tags.empty() )
            endElement();
    }

    XmlWriter& XmlWriter::startElement( std::string const& name ) {
        ensureTagClosed();
        newlineIfNecessary();
        m_os « m_indent « '<' « name;
        m_tags.push_back( name );
        m_indent += "  ";
        m_tagIsOpen = true;
        return *this;
    }

    XmlWriter::ScopedElement XmlWriter::scopedElement( std::string const& name ) {
        ScopedElement scoped( this );
        startElement( name );
        return scoped;
    }

    XmlWriter& XmlWriter::endElement() {
        newlineIfNecessary();
        m_indent = m_indent.substr( 0, m_indent.size()-2 );
        if( m_tagIsOpen ) {
            m_os « "/>";
            m_tagIsOpen = false;
        }
        else {
            m_os « m_indent « "</" « m_tags.back() « ">";
        }
        m_os « std::endl;
        m_tags.pop_back();
        return *this;
    }

    XmlWriter& XmlWriter::writeAttribute( std::string const& name, std::string const& attribute ) {
        if( !name.empty() && !attribute.empty() )
            m_os « ' ' « name « "=\"" « XmlEncode( attribute, XmlEncode::ForAttributes ) « '"';
        return *this;
    }

    XmlWriter& XmlWriter::writeAttribute( std::string const& name, const char* attribute ) {
        if( !name.empty() && attribute && attribute[0] != '\0' )
            m_os « ' ' « name « "=\"" « XmlEncode( attribute, XmlEncode::ForAttributes ) « '"';
        return *this;
    }

    XmlWriter& XmlWriter::writeAttribute( std::string const& name, bool attribute ) {
        m_os « ' ' « name « "=\"" « ( attribute ? "true" : "false" ) « '"';
        return *this;
    }

    XmlWriter& XmlWriter::writeText( std::string const& text, bool indent ) {
        if( !text.empty() ){
            bool tagWasOpen = m_tagIsOpen;
            ensureTagClosed();
            if( tagWasOpen && indent )
                m_os « m_indent;
            m_os « XmlEncode( text );
            m_needsNewline = true;
        }
        return *this;
    }

    //XmlWriter& XmlWriter::writeComment( std::string const& text ) {
    //    ensureTagClosed();
    //    m_os « m_indent « "<!--" « text « "-->";
    //    m_needsNewline = true;
    //    return *this;
    //}

    //void XmlWriter::writeStylesheetRef( std::string const& url ) {
    //    m_os « "<?xml-stylesheet type=\"text/xsl\" href=\"" « url « "\"?>\n";
    //}

    //XmlWriter& XmlWriter::writeBlankLine() {
    //    ensureTagClosed();
    //    m_os « '\n';
    //    return *this;
    //}

    void XmlWriter::ensureTagClosed() {
        if( m_tagIsOpen ) {
            m_os « ">" « std::endl;
            m_tagIsOpen = false;
        }
    }

    void XmlWriter::writeDeclaration() {
        m_os « "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
    }

    void XmlWriter::newlineIfNecessary() {
        if( m_needsNewline ) {
            m_os « std::endl;
            m_needsNewline = false;
        }
    }

```

```
// =================================================================================================
// End of copy-pasted code from Catch
// =================================================================================================

    // clang-format on

    struct XmlReporter : public IReporter
    {
        XmlWriter xml;
        DOCTEST_DECLARE_MUTEX(mutex)

        // caching pointers/references to objects of these types - safe to do
        const ContextOptions& opt;
        const TestCaseData*   tc = nullptr;

        XmlReporter(const ContextOptions& co)
                : xml(*co.cout)
                , opt(co) {}

        void log_contexts() {
            int num_contexts = get_num_active_contexts();
            if(num_contexts) {
                auto              contexts = get_active_contexts();
                std::stringstream ss;
                for(int i = 0; i < num_contexts; ++i) {
                    contexts[i]->stringify(&ss);
                    xml.scopedElement("Info").writeText(ss.str());
                    ss.str("");
                }
            }
        }

        unsigned line(unsigned l) const { return opt.no_line_numbers ? 0 : l; }

        void test_case_start_impl(const TestCaseData& in) {
            bool open_ts_tag = false;
            if(tc != nullptr) { // we have already opened a test suite
                if(std::strcmp(tc->m_test_suite, in.m_test_suite) != 0) {
                    xml.endElement();
                    open_ts_tag = true;
                }
            }
            else {
                open_ts_tag = true; // first test case ==> first test suite
            }

            if(open_ts_tag) {
                xml.startElement("TestSuite");
                xml.writeAttribute("name", in.m_test_suite);
            }

            tc = &in;
            xml.startElement("TestCase")
                    .writeAttribute("name", in.m_name)
                    .writeAttribute("filename", skipPathFromFilename(in.m_file.c_str()))
                    .writeAttribute("line", line(in.m_line))
                    .writeAttribute("description", in.m_description);

            if(Approx(in.m_timeout) != 0)
                xml.writeAttribute("timeout", in.m_timeout);
            if(in.m_may_fail)
                xml.writeAttribute("may_fail", true);
            if(in.m_should_fail)
                xml.writeAttribute("should_fail", true);
        }

        // =========================================================================================
        // WHAT FOLLOWS ARE OVERRIDES OF THE VIRTUAL METHODS OF THE REPORTER INTERFACE
        // =========================================================================================

        void report_query(const QueryData& in) override {
            test_run_start();
            if(opt.list_reporters) {
                for(auto& curr : getListeners())
                    xml.scopedElement("Listener")
                            .writeAttribute("priority", curr.first.first)
                            .writeAttribute("name", curr.first.second);
                for(auto& curr : getReporters())
                    xml.scopedElement("Reporter")
                            .writeAttribute("priority", curr.first.first)
                            .writeAttribute("name", curr.first.second);
            } else if(opt.count || opt.list_test_cases) {
                for(unsigned i = 0; i < in.num_data; ++i) {
                    xml.scopedElement("TestCase").writeAttribute("name", in.data[i]->m_name)
                        .writeAttribute("testsuite", in.data[i]->m_test_suite)
                        .writeAttribute("filename", skipPathFromFilename(in.data[i]->m_file.c_str()))
                        .writeAttribute("line", line(in.data[i]->m_line))
                        .writeAttribute("skipped", in.data[i]->m_skip);
                }
                xml.scopedElement("OverallResultsTestCases")
                        .writeAttribute("unskipped", in.run_stats->numTestCasesPassingFilters);
            } else if(opt.list_test_suites) {
                for(unsigned i = 0; i < in.num_data; ++i)
                    xml.scopedElement("TestSuite").writeAttribute("name", in.data[i]->m_test_suite);
                xml.scopedElement("OverallResultsTestCases")
                        .writeAttribute("unskipped", in.run_stats->numTestCasesPassingFilters);
                xml.scopedElement("OverallResultsTestSuites")
                        .writeAttribute("unskipped", in.run_stats->numTestSuitesPassingFilters);
            }
            xml.endElement();
        }

        void test_run_start() override {
            xml.writeDeclaration();

            // remove .exe extension - mainly to have the same output on UNIX and Windows
            std::string binary_name = skipPathFromFilename(opt.binary_name.c_str());
#ifdef DOCTEST_PLATFORM_WINDOWS
            if(binary_name.rfind(".exe") != std::string::npos)
                binary_name = binary_name.substr(0, binary_name.length() - 4);
#endif // DOCTEST_PLATFORM_WINDOWS

            xml.startElement("doctest").writeAttribute("binary", binary_name);
            if(opt.no_version == false)
                xml.writeAttribute("version", DOCTEST_VERSION_STR);

            // only the consequential ones (TODO: filters)
            xml.scopedElement("Options")
                    .writeAttribute("order_by", opt.order_by.c_str())
                    .writeAttribute("rand_seed", opt.rand_seed)
                    .writeAttribute("first", opt.first)
                    .writeAttribute("last", opt.last)
                    .writeAttribute("abort_after", opt.abort_after)
                    .writeAttribute("subcase_filter_levels", opt.subcase_filter_levels)
                    .writeAttribute("case_sensitive", opt.case_sensitive)
                    .writeAttribute("no_throw", opt.no_throw)
                    .writeAttribute("no_skip", opt.no_skip);
        }

        void test_run_end(const TestRunStats& p) override {
            if(tc) // the TestSuite tag - only if there has been at least 1 test case
                xml.endElement();

            xml.scopedElement("OverallResultsAsserts")
                    .writeAttribute("successes", p.numAsserts - p.numAssertsFailed)
                    .writeAttribute("failures", p.numAssertsFailed);

            xml.startElement("OverallResultsTestCases")
                    .writeAttribute("successes",
                                    p.numTestCasesPassingFilters - p.numTestCasesFailed)
                    .writeAttribute("failures", p.numTestCasesFailed);
            if(opt.no_skipped_summary == false)
                xml.writeAttribute("skipped", p.numTestCases - p.numTestCasesPassingFilters);
            xml.endElement();

            xml.endElement();
        }

        void test_case_start(const TestCaseData& in) override {
            test_case_start_impl(in);
            xml.ensureTagClosed();
        }

        void test_case_reenter(const TestCaseData&) override {}

        void test_case_end(const CurrentTestCaseStats& st) override {
            xml.startElement("OverallResultsAsserts")
                    .writeAttribute("successes",
                                    st.numAssertsCurrentTest - st.numAssertsFailedCurrentTest)
                    .writeAttribute("failures", st.numAssertsFailedCurrentTest)
                    .writeAttribute("test_case_success", st.testCaseSuccess);
            if(opt.duration)
                xml.writeAttribute("duration", st.seconds);
            if(tc->m_expected_failures)
                xml.writeAttribute("expected_failures", tc->m_expected_failures);
            xml.endElement();

            xml.endElement();
        }

        void test_case_exception(const TestCaseException& e) override {
            DOCTEST_LOCK_MUTEX(mutex)

            xml.scopedElement("Exception")
```

```
                    .writeAttribute("crash", e.is_crash)
                    .writeText(e.error_string.c_str());
        }

        void subcase_start(const SubcaseSignature& in) override {
            xml.startElement("SubCase")
                    .writeAttribute("name", in.m_name)
                    .writeAttribute("filename", skipPathFromFilename(in.m_file))
                    .writeAttribute("line", line(in.m_line));
            xml.ensureTagClosed();
        }

        void subcase_end() override { xml.endElement(); }

        void log_assert(const AssertData& rb) override {
            if(!rb.m_failed && !opt.success)
                return;

            DOCTEST_LOCK_MUTEX(mutex)

            xml.startElement("Expression")
                    .writeAttribute("success", !rb.m_failed)
                    .writeAttribute("type", assertString(rb.m_at))
                    .writeAttribute("filename", skipPathFromFilename(rb.m_file))
                    .writeAttribute("line", line(rb.m_line));

            xml.scopedElement("Original").writeText(rb.m_expr);

            if(rb.m_threw)
                xml.scopedElement("Exception").writeText(rb.m_exception.c_str());

            if(rb.m_at & assertType::is_throws_as)
                xml.scopedElement("ExpectedException").writeText(rb.m_exception_type);
            if(rb.m_at & assertType::is_throws_with)
                xml.scopedElement("ExpectedExceptionString").writeText(rb.m_exception_string.c_str());
            if((rb.m_at & assertType::is_normal) && !rb.m_threw)
                xml.scopedElement("Expanded").writeText(rb.m_decomp.c_str());

            log_contexts();

            xml.endElement();
        }

        void log_message(const MessageData& mb) override {
            DOCTEST_LOCK_MUTEX(mutex)

            xml.startElement("Message")
                    .writeAttribute("type", failureString(mb.m_severity))
                    .writeAttribute("filename", skipPathFromFilename(mb.m_file))
                    .writeAttribute("line", line(mb.m_line));

            xml.scopedElement("Text").writeText(mb.m_string.c_str());

            log_contexts();

            xml.endElement();
        }

        void test_case_skipped(const TestCaseData& in) override {
            if(opt.no_skipped_summary == false) {
                test_case_start_impl(in);
                xml.writeAttribute("skipped", "true");
                xml.endElement();
            }
        }
    };

    DOCTEST_REGISTER_REPORTER("xml", 0, XmlReporter);

    void fulltext_log_assert_to_stream(std::ostream& s, const AssertData& rb) {
        if((rb.m_at & (assertType::is_throws_as | assertType::is_throws_with)) ==
            0)
            s « Color::Cyan « assertString(rb.m_at) « "( " « rb.m_expr « " ) "
                « Color::None;

        if(rb.m_at & assertType::is_throws) {
            s « (rb.m_threw ? "threw as expected!" : "did NOT throw at all!") « "\n";
        } else if((rb.m_at & assertType::is_throws_as) &&
                    (rb.m_at & assertType::is_throws_with)) {
            s « Color::Cyan « assertString(rb.m_at) « "( " « rb.m_expr « ", \""
                « rb.m_exception_string.c_str()
                « "\", " « rb.m_exception_type « " ) " « Color::None;
            if(rb.m_threw) {
                if(!rb.m_failed) {
                    s « "threw as expected!\n";
                } else {
                    s « "threw a DIFFERENT exception! (contents: " « rb.m_exception « ")\n";
                }
            } else {
                s « "did NOT throw at all!\n";
            }
        } else if(rb.m_at &
                  assertType::is_throws_as) {
            s « Color::Cyan « assertString(rb.m_at) « "( " « rb.m_expr « ", "
                « rb.m_exception_type « " ) " « Color::None
                « (rb.m_threw ? (rb.m_threw_as ? "threw as expected!" :
                                                 "threw a DIFFERENT exception: ") :
                                "did NOT throw at all!")
                « Color::Cyan « rb.m_exception « "\n";
        } else if(rb.m_at &
                  assertType::is_throws_with) {
            s « Color::Cyan « assertString(rb.m_at) « "( " « rb.m_expr « ", \""
                « rb.m_exception_string.c_str()
                « "\" ) " « Color::None
                « (rb.m_threw ? (!rb.m_failed ? "threw as expected!" :
                                                 "threw a DIFFERENT exception: ") :
                                "did NOT throw at all!")
                « Color::Cyan « rb.m_exception « "\n";
        } else if(rb.m_at & assertType::is_nothrow) {
            s « (rb.m_threw ? "THREW exception: " : "didn't throw!") « Color::Cyan
                « rb.m_exception « "\n";
        } else {
            s « (rb.m_threw ? "THREW exception: " :
                                (!rb.m_failed ? "is correct!\n" : "is NOT correct!\n"));
            if(rb.m_threw)
                s « rb.m_exception « "\n";
            else
                s « "  values: " « assertString(rb.m_at) « "( " « rb.m_decomp « " )\n";
        }
    }

    // TODO:
    // - log_message()
    // - respond to queries
    // - honor remaining options
    // - more attributes in tags
    struct JUnitReporter : public IReporter
    {
        XmlWriter xml;
        DOCTEST_DECLARE_MUTEX(mutex)
        Timer timer;
        std::vector<String> deepestSubcaseStackNames;

        struct JUnitTestCaseData
        {
            static std::string getCurrentTimestamp() {
                // Beware, this is not reentrant because of backward compatibility issues
                // Also, UTC only, again because of backward compatibility (%z is C++11)
                time_t rawtime;
                std::time(&rawtime);
                auto const timeStampSize = sizeof("2017-01-16T17:06:45Z");

                std::tm timeInfo;
#ifdef DOCTEST_PLATFORM_WINDOWS
                gmtime_s(&timeInfo, &rawtime);
#else // DOCTEST_PLATFORM_WINDOWS
                gmtime_r(&rawtime, &timeInfo);
#endif // DOCTEST_PLATFORM_WINDOWS

                char timeStamp[timeStampSize];
                const char* const fmt = "%Y-%m-%dT%H:%M:%SZ";

                std::strftime(timeStamp, timeStampSize, fmt, &timeInfo);
                return std::string(timeStamp);
            }

            struct JUnitTestMessage
            {
                JUnitTestMessage(const std::string& _message, const std::string& _type, const
std::string& _details)
                    : message(_message), type(_type), details(_details) {}

                JUnitTestMessage(const std::string& _message, const std::string& _details)
                    : message(_message), type(), details(_details) {}

                std::string message, type, details;
            };

            struct JUnitTestCase
            {
                JUnitTestCase(const std::string& _classname, const std::string& _name)
                    : classname(_classname), name(_name), time(0), failures() {}

                std::string classname, name;
                double time;
                std::vector<JUnitTestMessage> failures, errors;
            };

            void add(const std::string& classname, const std::string& name) {
                testcases.emplace_back(classname, name);
            }

            void appendSubcaseNamesToLastTestcase(std::vector<String> nameStack) {
                for(auto& curr: nameStack)
                    if(curr.size())
                        testcases.back().name += std::string("/") + curr.c_str();
            }

            void addTime(double time) {
                if(time < 1e-4)
                    time = 0;
                testcases.back().time = time;
                totalSeconds += time;
            }

            void addFailure(const std::string& message, const std::string& type, const std::string&
details) {
                testcases.back().failures.emplace_back(message, type, details);
                ++totalFailures;
            }

            void addError(const std::string& message, const std::string& details) {
                testcases.back().errors.emplace_back(message, details);
                ++totalErrors;
            }

            std::vector<JUnitTestCase> testcases;
            double totalSeconds = 0;
            int totalErrors = 0, totalFailures = 0;
        };

        JUnitTestCaseData testCaseData;

        // caching pointers/references to objects of these types - safe to do
        const ContextOptions& opt;
        const TestCaseData*   tc = nullptr;

        JUnitReporter(const ContextOptions& co)
                : xml(*co.cout)
                , opt(co) {}

        unsigned line(unsigned l) const { return opt.no_line_numbers ? 0 : l; }

        // =========================================================================================
        // WHAT FOLLOWS ARE OVERRIDES OF THE VIRTUAL METHODS OF THE REPORTER INTERFACE
        // =========================================================================================

        void report_query(const QueryData&) override {
            xml.writeDeclaration();
        }

        void test_run_start() override {
            xml.writeDeclaration();
        }

        void test_run_end(const TestRunStats& p) override {
            // remove .exe extension - mainly to have the same output on UNIX and Windows
            std::string binary_name = skipPathFromFilename(opt.binary_name.c_str());
#ifdef DOCTEST_PLATFORM_WINDOWS
            if(binary_name.rfind(".exe") != std::string::npos)
                binary_name = binary_name.substr(0, binary_name.length() - 4);
#endif // DOCTEST_PLATFORM_WINDOWS
            xml.startElement("testsuites");
            xml.startElement("testsuite").writeAttribute("name", binary_name)
                    .writeAttribute("errors", testCaseData.totalErrors)
                    .writeAttribute("failures", testCaseData.totalFailures)
                    .writeAttribute("tests", p.numAsserts);
            if(opt.no_time_in_output == false) {
                xml.writeAttribute("time", testCaseData.totalSeconds);
                xml.writeAttribute("timestamp", JUnitTestCaseData::getCurrentTimestamp());
            }
            if(opt.no_version == false)
                xml.writeAttribute("doctest_version", DOCTEST_VERSION_STR);

            for(const auto& testCase : testCaseData.testcases) {
                xml.startElement("testcase")
                    .writeAttribute("classname", testCase.classname)
                    .writeAttribute("name", testCase.name);
                if(opt.no_time_in_output == false)
                    xml.writeAttribute("time", testCase.time);
                // This is not ideal, but it should be enough to mimic gtest's junit output.
                xml.writeAttribute("status", "run");

                for(const auto& failure : testCase.failures) {
                    xml.scopedElement("failure")
                        .writeAttribute("message", failure.message)
                        .writeAttribute("type", failure.type)
                        .writeText(failure.details, false);
                }

                for(const auto& error : testCase.errors) {
                    xml.scopedElement("error")
                        .writeAttribute("message", error.message)
                        .writeText(error.details);
                }

                xml.endElement();
            }
            xml.endElement();
            xml.endElement();
        }

        void test_case_start(const TestCaseData& in) override {
            testCaseData.add(skipPathFromFilename(in.m_file.c_str()), in.m_name);
            timer.start();
        }

        void test_case_reenter(const TestCaseData& in) override {
            testCaseData.addTime(timer.getElapsedSeconds());
            testCaseData.appendSubcaseNamesToLastTestcase(deepestSubcaseStackNames);
            deepestSubcaseStackNames.clear();

            timer.start();
            testCaseData.add(skipPathFromFilename(in.m_file.c_str()), in.m_name);
        }

        void test_case_end(const CurrentTestCaseStats&) override {
            testCaseData.addTime(timer.getElapsedSeconds());
            testCaseData.appendSubcaseNamesToLastTestcase(deepestSubcaseStackNames);
            deepestSubcaseStackNames.clear();
        }

        void test_case_exception(const TestCaseException& e) override {
            DOCTEST_LOCK_MUTEX(mutex)
            testCaseData.addError("exception", e.error_string.c_str());
        }

        void subcase_start(const SubcaseSignature& in) override {
            deepestSubcaseStackNames.push_back(in.m_name);
        }

        void subcase_end() override {}

        void log_assert(const AssertData& rb) override {
            if(!rb.m_failed) // report only failures & ignore the `success` option
                return;

            DOCTEST_LOCK_MUTEX(mutex)

            std::ostringstream os;
            os « skipPathFromFilename(rb.m_file) « (opt.gnu_file_line ? ":" : "(")
              « line(rb.m_line) « (opt.gnu_file_line ? ":" : "):") « std::endl;

            fulltext_log_assert_to_stream(os, rb);
            log_contexts(os);
            testCaseData.addFailure(rb.m_decomp.c_str(), assertString(rb.m_at), os.str());
        }

        void log_message(const MessageData& mb) override {
            if(mb.m_severity & assertType::is_warn) // report only failures
                return;

            DOCTEST_LOCK_MUTEX(mutex)

            std::ostringstream os;
            os « skipPathFromFilename(mb.m_file) « (opt.gnu_file_line ? ":" : "(")
              « line(mb.m_line) « (opt.gnu_file_line ? ":" : "):") « std::endl;

            os « mb.m_string.c_str() « "\n";
            log_contexts(os);

            testCaseData.addFailure(mb.m_string.c_str(),
                mb.m_severity & assertType::is_check ? "FAIL_CHECK" : "FAIL", os.str());
        }

        void test_case_skipped(const TestCaseData&) override {}

        void log_contexts(std::ostringstream& s) {
            int num_contexts = get_num_active_contexts();
            if(num_contexts) {
                auto contexts = get_active_contexts();

                s « "  logged: ";
                for(int i = 0; i < num_contexts; ++i) {
                    s « (i == 0 ? "" : "          ");
                    contexts[i]->stringify(&s);
                    s « std::endl;
                }
            }
        }
    };

    DOCTEST_REGISTER_REPORTER("junit", 0, JUnitReporter);

    struct Whitespace
    {
        int nrSpaces;
        explicit Whitespace(int nr)
                : nrSpaces(nr) {}
    };

    std::ostream& operator«(std::ostream& out, const Whitespace& ws) {
        if(ws.nrSpaces != 0)
            out « std::setw(ws.nrSpaces) « ' ';
        return out;
    }

    struct ConsoleReporter : public IReporter
    {
        std::ostream&                 s;
        bool                          hasLoggedCurrentTestStart;
        std::vector<SubcaseSignature> subcasesStack;
        size_t                        currentSubcaseLevel;
        DOCTEST_DECLARE_MUTEX(mutex)

        // caching pointers/references to objects of these types - safe to do
        const ContextOptions& opt;
        const TestCaseData*   tc;

        ConsoleReporter(const ContextOptions& co)
                : s(*co.cout)
                , opt(co) {}

        ConsoleReporter(const ContextOptions& co, std::ostream& ostr)
                : s(ostr)
                , opt(co) {}

        // =========================================================================================
        // WHAT FOLLOWS ARE HELPERS USED BY THE OVERRIDES OF THE VIRTUAL METHODS OF THE INTERFACE
        // =========================================================================================

        void separator_to_stream() {
            s « Color::Yellow
              « "==============================================================================="
                "\n";
        }

        const char* getSuccessOrFailString(bool success, assertType::Enum at,
                                           const char* success_str) {
            if(success)
                return success_str;
            return failureString(at);
        }

        Color::Enum getSuccessOrFailColor(bool success, assertType::Enum at) {
            return success ? Color::BrightGreen :
                             (at & assertType::is_warn) ? Color::Yellow : Color::Red;
        }

        void successOrFailColoredStringToStream(bool success, assertType::Enum at,
                                                const char* success_str = "SUCCESS") {
            s « getSuccessOrFailColor(success, at)
              « getSuccessOrFailString(success, at, success_str) « ": ";
        }

        void log_contexts() {
            int num_contexts = get_num_active_contexts();
            if(num_contexts) {
                auto contexts = get_active_contexts();

                s « Color::None « "  logged: ";
                for(int i = 0; i < num_contexts; ++i) {
                    s « (i == 0 ? "" : "          ");
                    contexts[i]->stringify(&s);
                    s « "\n";
                }
            }

            s « "\n";
        }

        // this was requested to be made virtual so users could override it
        virtual void file_line_to_stream(const char* file, int line,
                                        const char* tail = "") {
            s « Color::LightGrey « skipPathFromFilename(file) « (opt.gnu_file_line ? ":" : "(")
            « (opt.no_line_numbers ? 0 : line) // 0 or the real num depending on the option
            « (opt.gnu_file_line ? ":" : "):") « tail;
        }

        void logTestStart() {
            if(hasLoggedCurrentTestStart)
                return;

            separator_to_stream();
            file_line_to_stream(tc->m_file.c_str(), tc->m_line, "\n");
            if(tc->m_description)
                s « Color::Yellow « "DESCRIPTION: " « Color::None « tc->m_description « "\n";
            if(tc->m_test_suite && tc->m_test_suite[0] != '\0')
                s « Color::Yellow « "TEST SUITE: " « Color::None « tc->m_test_suite « "\n";
            if(strncmp(tc->m_name, "  Scenario:", 11) != 0)
                s « Color::Yellow « "TEST CASE:  ";
            s « Color::None « tc->m_name « "\n";

            for(size_t i = 0; i < currentSubcaseLevel; ++i) {
                if(subcasesStack[i].m_name[0] != '\0')
                    s « "  " « subcasesStack[i].m_name « "\n";
            }

            if(currentSubcaseLevel != subcasesStack.size()) {
                s « Color::Yellow « "\nDEEPEST SUBCASE STACK REACHED (DIFFERENT FROM THE CURRENT ONE):\n" « Color::None;
                for(size_t i = 0; i < subcasesStack.size(); ++i) {
                    if(subcasesStack[i].m_name[0] != '\0')
                        s « "  " « subcasesStack[i].m_name « "\n";
                }
            }

            s « "\n";

            hasLoggedCurrentTestStart = true;
        }

        void printVersion() {
            if(opt.no_version == false)
                s « Color::Cyan « "[doctest] " « Color::None « "doctest version is \""
                  « DOCTEST_VERSION_STR « "\"\n";
        }

        void printIntro() {
            if(opt.no_intro == false) {
                printVersion();
                s « Color::Cyan « "[doctest] " « Color::None
                  « "run with \"--" DOCTEST_OPTIONS_PREFIX_DISPLAY "help\" for options\n";
            }
        }

        void printHelp() {
            int sizePrefixDisplay = static_cast<int>(strlen(DOCTEST_OPTIONS_PREFIX_DISPLAY));
            printVersion();
            // clang-format off
            s « Color::Cyan « "[doctest]\n" « Color::None;
            s « Color::Cyan « "[doctest] " « Color::None;
            s « "boolean values: \"1/on/yes/true\" or \"0/off/no/false\"\n";
            s « Color::Cyan « "[doctest] " « Color::None;
            s « "filter  values: \"str1,str2,str3\" (comma separated strings)\n";
            s « Color::Cyan « "[doctest]\n" « Color::None;
            s « Color::Cyan « "[doctest] " « Color::None;
            s « "filters use wildcards for matching strings\n";
            s « Color::Cyan « "[doctest] " « Color::None;
            s « "something passes a filter if any of the strings in a filter matches\n";
#ifndef DOCTEST_CONFIG_NO_UNPREFIXED_OPTIONS
            s « Color::Cyan « "[doctest]\n" « Color::None;
            s « Color::Cyan « "[doctest] " « Color::None;
            s « "ALL FLAGS, OPTIONS AND FILTERS ALSO AVAILABLE WITH A \"" DOCTEST_CONFIG_OPTIONS_PREFIX "\" PREFIX!!!\n";
#endif
            s « Color::Cyan « "[doctest]\n" « Color::None;
            s « Color::Cyan « "[doctest] " « Color::None;
            s « "Query flags - the program quits after them. Available:\n\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "?,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY "help, -" DOCTEST_OPTIONS_PREFIX_DISPLAY "h                      "
              « Whitespace(sizePrefixDisplay*0) «  "prints this message\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "v,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY "version
"
              « Whitespace(sizePrefixDisplay*1) « "prints the version\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "c,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY "count
"
              « Whitespace(sizePrefixDisplay*1) « "prints the number of matching tests\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "ltc, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"list-test-cases              "
              « Whitespace(sizePrefixDisplay*1) « "lists all matching tests by name\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "lts, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"list-test-suites             "
              « Whitespace(sizePrefixDisplay*1) « "lists all matching test suites\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "lr,  --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"list-reporters               "
              « Whitespace(sizePrefixDisplay*1) « "lists all registered reporters\n\n";
            // ================================================================================== « 79
            s « Color::Cyan « "[doctest] " « Color::None;
            s « "The available <int>/<string> options/filters are:\n\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "tc,  --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"test-case=<filters>          "
              « Whitespace(sizePrefixDisplay*1) « "filters     tests by their name\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "tce, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"test-case-exclude=<filters>  "
              « Whitespace(sizePrefixDisplay*1) « "filters OUT tests by their name\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "sf,  --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"source-file=<filters>        "
              « Whitespace(sizePrefixDisplay*1) « "filters     tests by their file\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "sfe, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"source-file-exclude=<filters> "
              « Whitespace(sizePrefixDisplay*1) « "filters OUT tests by their file\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "ts,  --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"test-suite=<filters>         "
              « Whitespace(sizePrefixDisplay*1) « "filters     tests by their test suite\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "tse, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"test-suite-exclude=<filters>  "
              « Whitespace(sizePrefixDisplay*1) « "filters OUT tests by their test suite\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "sc,  --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"subcase=<filters>            "
              « Whitespace(sizePrefixDisplay*1) « "filters     subcases by their name\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "sce, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"subcase-exclude=<filters>    "
              « Whitespace(sizePrefixDisplay*1) « "filters OUT subcases by their name\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "r,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"reporters=<filters>          "
              « Whitespace(sizePrefixDisplay*1) « "reporters to use (console is default)\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "o,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"out=<string>                 "
              « Whitespace(sizePrefixDisplay*1) « "output filename\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "ob,  --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"order-by=<string>            "
              « Whitespace(sizePrefixDisplay*1) « "how the tests should be ordered\n";
            s « Whitespace(sizePrefixDisplay*3) « "                                       <string> -
[file/suite/name/rand/none]\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "rs,  --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"rand-seed=<int>              "
              « Whitespace(sizePrefixDisplay*1) « "seed for random ordering\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "f,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"first=<int>                  "
              « Whitespace(sizePrefixDisplay*1) « "the first test passing the filters to\n";
            s « Whitespace(sizePrefixDisplay*3) « "                                       execute -
for range-based execution\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "l,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"last=<int>                   "
              « Whitespace(sizePrefixDisplay*1) « "the last test passing the filters to\n";
            s « Whitespace(sizePrefixDisplay*3) « "                                       execute -
for range-based execution\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "aa,  --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"abort-after=<int>            "
              « Whitespace(sizePrefixDisplay*1) « "stop after <int> failed assertions\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "scfl,--" DOCTEST_OPTIONS_PREFIX_DISPLAY
"subcase-filter-levels=<int>  "
              « Whitespace(sizePrefixDisplay*1) « "apply filters for the first <int> levels\n";
            s « Color::Cyan « "\n[doctest] " « Color::None;
            s « "Bool options - can be used like flags and true is assumed. Available:\n\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "s,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"success=<bool>               "
              « Whitespace(sizePrefixDisplay*1) « "include successful assertions in output\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "cs,  --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"case-sensitive=<bool>        "
              « Whitespace(sizePrefixDisplay*1) « "filters being treated as case sensitive\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "e,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"exit=<bool>                  "
              « Whitespace(sizePrefixDisplay*1) « "exits after the tests finish\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "d,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"duration=<bool>              "
              « Whitespace(sizePrefixDisplay*1) « "prints the time duration of each test\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "m,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
```

```
"minimal=<bool>                 "
              « Whitespace(sizePrefixDisplay*1) « "minimal console output (only failures)\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "q,    --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"quiet=<bool>                   "
              « Whitespace(sizePrefixDisplay*1) « "no console output\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "nt,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-throw=<bool>                "
              « Whitespace(sizePrefixDisplay*1) « "skips exceptions-related assert checks\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "ne,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-exitcode=<bool>             "
              « Whitespace(sizePrefixDisplay*1) « "returns (or exits) always with success\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "nr,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-run=<bool>                  "
              « Whitespace(sizePrefixDisplay*1) « "skips all runtime doctest operations\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "ni,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-intro=<bool>                "
              « Whitespace(sizePrefixDisplay*1) « "omit the framework intro in the output\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "nv,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-version=<bool>              "
              « Whitespace(sizePrefixDisplay*1) « "omit the framework version in the output\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "nc,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-colors=<bool>               "
              « Whitespace(sizePrefixDisplay*1) « "disables colors in output\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "fc,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"force-colors=<bool>            "
              « Whitespace(sizePrefixDisplay*1) « "use colors even when not in a tty\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "nb,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-breaks=<bool>               "
              « Whitespace(sizePrefixDisplay*1) « "disables breakpoints in debuggers\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "ns,   --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-skip=<bool>                 "
              « Whitespace(sizePrefixDisplay*1) « "don’t skip test cases marked as skip\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "gfl, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"gnu-file-line=<bool>           "
              « Whitespace(sizePrefixDisplay*1) « ":n: vs (n): for line numbers in output\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "npf, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-path-filenames=<bool>       "
              « Whitespace(sizePrefixDisplay*1) « "only filenames and no paths in output\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "spp, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"skip-path-prefixes=<p1:p2>     "
              « Whitespace(sizePrefixDisplay*1) « "whenever file paths start with this prefix, remove
it from the output\n";
            s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "nln, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-line-numbers=<bool>         "
              « Whitespace(sizePrefixDisplay*1) « "0 instead of real line numbers in output\n";
            // ================================================================================== « 79
            // clang-format on

            s « Color::Cyan « "\n[doctest] " « Color::None;
            s « "for more information visit the project documentation\n\n";
        }

        void printRegisteredReporters() {
            printVersion();
            auto printReporters = [this] (const reporterMap& reporters, const char* type) {
                if(reporters.size()) {
                    s « Color::Cyan « "[doctest] " « Color::None « "listing all registered " « type «
"\n";
                    for(auto& curr : reporters)
                        s « "priority: " « std::setw(5) « curr.first.first
                          « " name: " « curr.first.second « "\n";
                }
            };
            printReporters(getListeners(), "listeners");
            printReporters(getReporters(), "reporters");
        }

        // =========================================================================================
        // WHAT FOLLOWS ARE OVERRIDES OF THE VIRTUAL METHODS OF THE REPORTER INTERFACE
        // =========================================================================================

        void report_query(const QueryData& in) override {
            if(opt.version) {
                printVersion();
            } else if(opt.help) {
                printHelp();
            } else if(opt.list_reporters) {
                printRegisteredReporters();
            } else if(opt.count || opt.list_test_cases) {
                if(opt.list_test_cases) {
                    s « Color::Cyan « "[doctest] " « Color::None
                      « "listing all test case names\n";
                    separator_to_stream();
                }

                for(unsigned i = 0; i < in.num_data; ++i)
                    s « Color::None « in.data[i]->m_name « "\n";

                separator_to_stream();

                s « Color::Cyan « "[doctest] " « Color::None
                  « "unskipped test cases passing the current filters: "
                  « g_cs->numTestCasesPassingFilters « "\n";

            } else if(opt.list_test_suites) {
                s « Color::Cyan « "[doctest] " « Color::None « "listing all test suites\n";
                separator_to_stream();

                for(unsigned i = 0; i < in.num_data; ++i)
                    s « Color::None « in.data[i]->m_test_suite « "\n";

                separator_to_stream();

                s « Color::Cyan « "[doctest] " « Color::None
                  « "unskipped test cases passing the current filters: "
                  « g_cs->numTestCasesPassingFilters « "\n";
                s « Color::Cyan « "[doctest] " « Color::None
                  « "test suites with unskipped test cases passing the current filters: "
                  « g_cs->numTestSuitesPassingFilters « "\n";
            }
        }

        void test_run_start() override {
            if(!opt.minimal)
                printIntro();
        }

        void test_run_end(const TestRunStats& p) override {
            if(opt.minimal && p.numTestCasesFailed == 0)
                return;

            separator_to_stream();
            s « std::dec;

            auto totwidth =
int(std::ceil(log10(static_cast<double>(std::max(p.numTestCasesPassingFilters,
static_cast<unsigned>(p.numAsserts))) + 1)));
            auto passwidth =
int(std::ceil(log10(static_cast<double>(std::max(p.numTestCasesPassingFilters - p.numTestCasesFailed,
static_cast<unsigned>(p.numAsserts - p.numAssertsFailed))) + 1)));
            auto failwidth = int(std::ceil(log10(static_cast<double>(std::max(p.numTestCasesFailed,
static_cast<unsigned>(p.numAssertsFailed))) + 1)));
            const bool anythingFailed = p.numTestCasesFailed > 0 || p.numAssertsFailed > 0;
            s « Color::Cyan « "[doctest] " « Color::None « "test cases: " « std::setw(totwidth)
              « p.numTestCasesPassingFilters « " | "
              « ((p.numTestCasesPassingFilters == 0 || anythingFailed) ? Color::None :
                                                                          Color::Green)
              « std::setw(passwidth) « p.numTestCasesPassingFilters - p.numTestCasesFailed « " passed"
              « Color::None « " | " « (p.numTestCasesFailed > 0 ? Color::Red : Color::None)
              « std::setw(failwidth) « p.numTestCasesFailed « " failed" « Color::None « " |";
            if(opt.no_skipped_summary == false) {
                const int numSkipped = p.numTestCases - p.numTestCasesPassingFilters;
                s « " " « (numSkipped == 0 ? Color::None : Color::Yellow) « numSkipped
                  « " skipped" « Color::None;
            }
            s « "\n";
            s « Color::Cyan « "[doctest] " « Color::None « "assertions: " « std::setw(totwidth)
              « p.numAsserts « " | "
              « ((p.numAsserts == 0 || anythingFailed) ? Color::None : Color::Green)
              « std::setw(passwidth) « (p.numAsserts - p.numAssertsFailed) « " passed" « Color::None
              « " | " « (p.numAssertsFailed > 0 ? Color::Red : Color::None) « std::setw(failwidth)
              « p.numAssertsFailed « " failed" « Color::None « " |\n";
            s « Color::Cyan « "[doctest] " « Color::None
              « "Status: " « (p.numTestCasesFailed > 0 ? Color::Red : Color::Green)
              « ((p.numTestCasesFailed > 0) ? "FAILURE!" : "SUCCESS!") « Color::None « std::endl;
        }

        void test_case_start(const TestCaseData& in) override {
            hasLoggedCurrentTestStart = false;
            tc                        = &in;
            subcasesStack.clear();
            currentSubcaseLevel = 0;
        }

        void test_case_reenter(const TestCaseData&) override {
            subcasesStack.clear();
        }

        void test_case_end(const CurrentTestCaseStats& st) override {
            if(tc->m_no_output)
                return;

            // log the preamble of the test case only if there is something
            // else to print - something other than that an assert has failed
            if(opt.duration ||
               (st.failure_flags && st.failure_flags !=
static_cast<int>(TestCaseFailureReason::AssertFailure)))
                logTestStart();

            if(opt.duration)
                s « Color::None « std::setprecision(6) « std::fixed « st.seconds
                  « " s: " « tc->m_name « "\n";

            if(st.failure_flags & TestCaseFailureReason::Timeout)
                s « Color::Red « "Test case exceeded time limit of " « std::setprecision(6)
                  « std::fixed « tc->m_timeout « "!\n";

            if(st.failure_flags & TestCaseFailureReason::ShouldHaveFailedButDidnt) {
                s « Color::Red « "Should have failed but didn't! Marking it as failed!\n";
            } else if(st.failure_flags & TestCaseFailureReason::ShouldHaveFailedAndDid) {
                s « Color::Yellow « "Failed as expected so marking it as not failed\n";
            } else if(st.failure_flags & TestCaseFailureReason::CouldHaveFailedAndDid) {
                s « Color::Yellow « "Allowed to fail so marking it as not failed\n";
            } else if(st.failure_flags & TestCaseFailureReason::DidntFailExactlyNumTimes) {
                s « Color::Red « "Didn't fail exactly " « tc->m_expected_failures
                  « " times so marking it as failed!\n";
            } else if(st.failure_flags & TestCaseFailureReason::FailedExactlyNumTimes) {
                s « Color::Yellow « "Failed exactly " « tc->m_expected_failures
                  « " times as expected so marking it as not failed!\n";
            }
            if(st.failure_flags & TestCaseFailureReason::TooManyFailedAsserts) {
                s « Color::Red « "Aborting - too many failed asserts!\n";
            }
            s « Color::None; // lgtm [cpp/useless-expression]
        }

        void test_case_exception(const TestCaseException& e) override {
            DOCTEST_LOCK_MUTEX(mutex)
            if(tc->m_no_output)
                return;

            logTestStart();

            file_line_to_stream(tc->m_file.c_str(), tc->m_line, " ");
            successOrFailColoredStringToStream(false, e.is_crash ? assertType::is_require :
                                                                    assertType::is_check);
            s « Color::Red « (e.is_crash ? "test case CRASHED: " : "test case THREW exception: ")
              « Color::Cyan « e.error_string « "\n";

            int num_stringified_contexts = get_num_stringified_contexts();
            if(num_stringified_contexts) {
                auto stringified_contexts = get_stringified_contexts();
                s « Color::None « "  logged: ";
                for(int i = num_stringified_contexts; i > 0; --i) {
                    s « (i == num_stringified_contexts ? "" : "          ")
                      « stringified_contexts[i - 1] « "\n";
                }
            }
            s « "\n" « Color::None;
        }

        void subcase_start(const SubcaseSignature& subc) override {
            subcasesStack.push_back(subc);
            ++currentSubcaseLevel;
            hasLoggedCurrentTestStart = false;
        }

        void subcase_end() override {
            --currentSubcaseLevel;
            hasLoggedCurrentTestStart = false;
        }

        void log_assert(const AssertData& rb) override {
            if((!rb.m_failed && !opt.success) || tc->m_no_output)
                return;

            DOCTEST_LOCK_MUTEX(mutex)

            logTestStart();

            file_line_to_stream(rb.m_file, rb.m_line, " ");
            successOrFailColoredStringToStream(!rb.m_failed, rb.m_at);

            fulltext_log_assert_to_stream(s, rb);

            log_contexts();
        }

        void log_message(const MessageData& mb) override {
            if(tc->m_no_output)
                return;

            DOCTEST_LOCK_MUTEX(mutex)

            logTestStart();

            file_line_to_stream(mb.m_file, mb.m_line, " ");
            s « getSuccessOrFailColor(false, mb.m_severity)
              « getSuccessOrFailString(mb.m_severity & assertType::is_warn, mb.m_severity,
                                        "MESSAGE") « ": ";
            s « Color::None « mb.m_string « "\n";
            log_contexts();
        }

        void test_case_skipped(const TestCaseData&) override {}
    };

    DOCTEST_REGISTER_REPORTER("console", 0, ConsoleReporter);

#ifdef DOCTEST_PLATFORM_WINDOWS
    struct DebugOutputWindowReporter : public ConsoleReporter
    {
        DOCTEST_THREAD_LOCAL static std::ostringstream oss;

        DebugOutputWindowReporter(const ContextOptions& co)
                : ConsoleReporter(co, oss) {}

#define DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(func, type, arg)                                    \
    void func(type arg) override {                                                                 \
        bool with_col = g_no_colors;                                                               \
        g_no_colors   = false;                                                                     \
        ConsoleReporter::func(arg);                                                                \
        if(oss.tellp() != std::streampos{}) {                                                      \
            DOCTEST_OUTPUT_DEBUG_STRING(oss.str().c_str());                                        \
            oss.str("");                                                                           \
        }                                                                                          \
        g_no_colors = with_col;                                                                    \
    }

        DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(test_run_start, DOCTEST_EMPTY, DOCTEST_EMPTY)
        DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(test_run_end, const TestRunStats&, in)
        DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(test_case_start, const TestCaseData&, in)
        DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(test_case_reenter, const TestCaseData&, in)
        DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(test_case_end, const CurrentTestCaseStats&, in)
        DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(test_case_exception, const TestCaseException&, in)
        DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(subcase_start, const SubcaseSignature&, in)
        DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(subcase_end, DOCTEST_EMPTY, DOCTEST_EMPTY)
        DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(log_assert, const AssertData&, in)
        DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(log_message, const MessageData&, in)
        DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(test_case_skipped, const TestCaseData&, in)
    };

    DOCTEST_THREAD_LOCAL std::ostringstream DebugOutputWindowReporter::oss;
#endif // DOCTEST_PLATFORM_WINDOWS

    // the implementation of parseOption()
    bool parseOptionImpl(int argc, const char* const* argv, const char* pattern, String* value) {
        // going from the end to the beginning and stopping on the first occurrence from the end
        for(int i = argc; i > 0; --i) {
            auto index = i - 1;
            auto temp = std::strstr(argv[index], pattern);
            if(temp && (value || strlen(temp) == strlen(pattern))) {
                // eliminate matches in which the chars before the option are not '-'
                bool noBadCharsFound = true;
                auto curr            = argv[index];
                while(curr != temp) {
                    if(*curr++ != '-') {
                        noBadCharsFound = false;
                        break;
                    }
                }
                if(noBadCharsFound && argv[index][0] == '-') {
                    if(value) {
                        // parsing the value of an option
                        temp += strlen(pattern);
                        const unsigned len = strlen(temp);
                        if(len) {
                            *value = temp;
                            return true;
                        }
                    } else {
                        // just a flag - no value
                        return true;
                    }
                }
            }
        }
        return false;
    }

    // parses an option and returns the string after the '=' character
    bool parseOption(int argc, const char* const* argv, const char* pattern, String* value = nullptr,
                     const String& defaultVal = String()) {
        if(value)
            *value = defaultVal;
#ifndef DOCTEST_CONFIG_NO_UNPREFIXED_OPTIONS
        // offset (normally 3 for "dt-") to skip prefix
        if(parseOptionImpl(argc, argv, pattern + strlen(DOCTEST_CONFIG_OPTIONS_PREFIX), value))
            return true;
#endif // DOCTEST_CONFIG_NO_UNPREFIXED_OPTIONS
        return parseOptionImpl(argc, argv, pattern, value);
    }

    // locates a flag on the command line
    bool parseFlag(int argc, const char* const* argv, const char* pattern) {
        return parseOption(argc, argv, pattern);
    }

    // parses a comma separated list of words after a pattern in one of the arguments in argv
    bool parseCommaSepArgs(int argc, const char* const* argv, const char* pattern,
                           std::vector<String>& res) {
        String filtersString;
        if(parseOption(argc, argv, pattern, &filtersString)) {
            // tokenize with "," as a separator, unless escaped with backslash
            std::ostringstream s;
            auto flush = [&s, &res]() {
                auto string = s.str();
                if(string.size() > 0) {
                    res.push_back(string.c_str());
                }
                s.str("");
            };

            bool seenBackslash = false;
            const char* current = filtersString.c_str();
            const char* end = current + strlen(current);
            while(current != end) {
                char character = *current++;
                if(seenBackslash) {
                    seenBackslash = false;
                    if(character == ',' || character == '\\') {
                        s.put(character);
                        continue;
                    }
                    s.put('\\');
                }
                if(character == '\\') {
                    seenBackslash = true;
                } else if(character == ',') {
                    flush();
                } else {
                    s.put(character);
                }
            }

            if(seenBackslash) {
                s.put('\\');
            }
            flush();
            return true;
        }
        return false;
    }

    enum optionType
    {
        option_bool,
        option_int
    };

    // parses an int/bool option from the command line
    bool parseIntOption(int argc, const char* const* argv, const char* pattern, optionType type,
                        int& res) {
        String parsedValue;
        if(!parseOption(argc, argv, pattern, &parsedValue))
            return false;

        if(type) {
            // integer
            // TODO: change this to use std::stoi or something else! currently it uses undefined
behavior - assumes '0' on failed parse...
            int theInt = std::atoi(parsedValue.c_str());
            if (theInt != 0) {
                res = theInt;
                return true;
            }
        } else {
            // boolean
            const char positive[][5] = { "1", "true", "on", "yes" };  // 5 - strlen("true") + 1
            const char negative[][6] = { "0", "false", "off", "no" }; // 6 - strlen("false") + 1

            // if the value matches any of the positive/negative possibilities
            for (unsigned i = 0; i < 4; i++) {
                if (parsedValue.compare(positive[i], true) == 0) {
                    res = 1;
                    return true;
                }
                if (parsedValue.compare(negative[i], true) == 0) {
                    res = 0;
                    return true;
                }
            }
        }
        return false;
    }
} // namespace

Context::Context(int argc, const char* const* argv)
        : p(new detail::ContextState) {
    parseArgs(argc, argv, true);
    if(argc)
        p->binary_name = argv[0];
}

Context::~Context() {
    if(g_cs == p)
        g_cs = nullptr;
    delete p;
}

void Context::applyCommandLine(int argc, const char* const* argv) {
    parseArgs(argc, argv);
    if(argc)
        p->binary_name = argv[0];
}

// parses args
void Context::parseArgs(int argc, const char* const* argv, bool withDefaults) {
    using namespace detail;

    // clang-format off
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "source-file=",        p->filters[0]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "sf=",                 p->filters[0]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "source-file-exclude=",p->filters[1]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "sfe=",                p->filters[1]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "test-suite=",         p->filters[2]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "ts=",                 p->filters[2]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "test-suite-exclude=", p->filters[3]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "tse=",                p->filters[3]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "test-case=",          p->filters[4]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "tc=",                 p->filters[4]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "test-case-exclude=",  p->filters[5]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "tce=",                p->filters[5]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "subcase=",            p->filters[6]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "sc=",                 p->filters[6]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "subcase-exclude=",    p->filters[7]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "sce=",                p->filters[7]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "reporters=",          p->filters[8]);
    parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "r=",                  p->filters[8]);
    // clang-format on

    int    intRes = 0;
    String strRes;

#define DOCTEST_PARSE_AS_BOOL_OR_FLAG(name, sname, var, default)                                  \
    if(parseIntOption(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX name "=", option_bool, intRes) ||  \
       parseIntOption(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX sname "=", option_bool, intRes))   \
        p->var = static_cast<bool>(intRes);                                                       \
    else if(parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX name) ||                          \
            parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX sname))                           \
        p->var = true;                                                                            \
    else if(withDefaults)                                                                         \
    p->var = default

#define DOCTEST_PARSE_INT_OPTION(name, sname, var, default)                                       \
    if(parseIntOption(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX name "=", option_int, intRes) ||   \
       parseIntOption(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX sname "=", option_int, intRes))    \
        p->var = intRes;                                                                          \
    else if(withDefaults)                                                                         \
    p->var = default

#define DOCTEST_PARSE_STR_OPTION(name, sname, var, default)                                       \
    if(parseOption(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX name "=", &strRes, default) ||       \
       parseOption(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX sname "=", &strRes, default) ||      \
       withDefaults)                                                                              \
    p->var = strRes

    // clang-format off
    DOCTEST_PARSE_STR_OPTION("out", "o", out, "");
    DOCTEST_PARSE_STR_OPTION("order-by", "ob", order_by, "file");
    DOCTEST_PARSE_INT_OPTION("rand-seed", "rs", rand_seed, 0);

    DOCTEST_PARSE_INT_OPTION("first", "f", first, 0);
    DOCTEST_PARSE_INT_OPTION("last", "l", last, UINT_MAX);

    DOCTEST_PARSE_INT_OPTION("abort-after", "aa", abort_after, 0);
    DOCTEST_PARSE_INT_OPTION("subcase-filter-levels", "scfl", subcase_filter_levels, INT_MAX);

    DOCTEST_PARSE_AS_BOOL_OR_FLAG("success", "s", success, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("case-sensitive", "cs", case_sensitive, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("exit", "e", exit, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("duration", "d", duration, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("minimal", "m", minimal, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("quiet", "q", quiet, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-throw", "nt", no_throw, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-exitcode", "ne", no_exitcode, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-run", "nr", no_run, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-intro", "ni", no_intro, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-version", "nv", no_version, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-colors", "nc", no_colors, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("force-colors", "fc", force_colors, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-breaks", "nb", no_breaks, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-skip", "ns", no_skip, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("gnu-file-line", "gfl", gnu_file_line, !bool(DOCTEST_MSVC));
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-path-filenames", "npf", no_path_in_filenames, false);
    DOCTEST_PARSE_STR_OPTION("strip-file-prefixes", "sfp", strip_file_prefixes, "");
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-line-numbers", "nln", no_line_numbers, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-debug-output", "ndo", no_debug_output, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-skipped-summary", "nss", no_skipped_summary, false);
    DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-time-in-output", "ntio", no_time_in_output, false);
    // clang-format on

    if(withDefaults) {
        p->help             = false;
        p->version          = false;
        p->count            = false;
        p->list_test_cases  = false;
        p->list_test_suites = false;
        p->list_reporters   = false;
    }
    if(parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "help") ||
       parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "h") ||
       parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "?")) {
        p->help = true;
        p->exit = true;
    }
    if(parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "version") ||
       parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "v")) {
        p->version = true;
        p->exit    = true;
    }
    if(parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "count") ||
       parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "c")) {
        p->count = true;
        p->exit  = true;
    }
    if(parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "list-test-cases") ||
       parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "ltc")) {
        p->list_test_cases = true;
        p->exit            = true;
    }
    if(parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "list-test-suites") ||
       parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "lts")) {
        p->list_test_suites = true;
        p->exit             = true;
    }
    if(parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "list-reporters") ||
       parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "lr")) {
        p->list_reporters = true;
        p->exit           = true;
    }
}

// allows the user to add procedurally to the filters from the command line
void Context::addFilter(const char* filter, const char* value) { setOption(filter, value); }

// allows the user to clear all filters from the command line
void Context::clearFilters() {
    for(auto& curr : p->filters)
```

```
        curr.clear();
}

// allows the user to override procedurally the bool options from the command line
void Context::setOption(const char* option, bool value) {
    setOption(option, value ? "true" : "false");
}

// allows the user to override procedurally the int options from the command line
void Context::setOption(const char* option, int value) {
    setOption(option, toString(value).c_str());
}

// allows the user to override procedurally the string options from the command line
void Context::setOption(const char* option, const char* value) {
    auto argv   = String("-") + option + "=" + value;
    auto lvalue = argv.c_str();
    parseArgs(1, &lvalue);
}

// users should query this in their main() and exit the program if true
bool Context::shouldExit() { return p->exit; }

void Context::setAsDefaultForAssertsOutOfTestCases() { g_cs = p; }

void Context::setAssertHandler(detail::assert_handler ah) { p->ah = ah; }

void Context::setCout(std::ostream* out) { p->cout = out; }

static class DiscardOStream : public std::ostream
{
private:
    class : public std::streambuf
    {
    private:
        // allowing some buffering decreases the amount of calls to overflow
        char buf[1024];

    protected:
        std::streamsize xsputn(const char_type*, std::streamsize count) override { return count; }

        int_type overflow(int_type ch) override {
            setp(std::begin(buf), std::end(buf));
            return traits_type::not_eof(ch);
        }
    } discardBuf;

public:
    DiscardOStream()
            : std::ostream(&discardBuf) {}
} discardOut;

// the main function that does all the filtering and test running
int Context::run() {
    using namespace detail;

    // save the old context state in case such was setup - for using asserts out of a testing context
    auto old_cs = g_cs;
    // this is the current contest
    g_cs               = p;
    is_running_in_test = true;

    g_no_colors = p->no_colors;
    p->resetRunData();

    std::fstream fstr;
    if(p->cout == nullptr) {
        if(p->quiet) {
            p->cout = &discardOut;
        } else if(p->out.size()) {
            // to a file if specified
            fstr.open(p->out.c_str(), std::fstream::out);
            p->cout = &fstr;
        } else {
#ifndef DOCTEST_CONFIG_NO_INCLUDE_IOSTREAM
            // stdout by default
            p->cout = &std::cout;
#else // DOCTEST_CONFIG_NO_INCLUDE_IOSTREAM
            return EXIT_FAILURE;
#endif // DOCTEST_CONFIG_NO_INCLUDE_IOSTREAM
        }
    }

    FatalConditionHandler::allocateAltStackMem();

    auto cleanup_and_return = [&]() {
        FatalConditionHandler::freeAltStackMem();

        if(fstr.is_open())
            fstr.close();

        // restore context
        g_cs               = old_cs;
        is_running_in_test = false;

        // we have to free the reporters which were allocated when the run started
        for(auto& curr : p->reporters_currently_used)
            delete curr;
        p->reporters_currently_used.clear();

        if(p->numTestCasesFailed && !p->no_exitcode)
            return EXIT_FAILURE;
        return EXIT_SUCCESS;
    };

    // setup default reporter if none is given through the command line
    if(p->filters[8].empty())
        p->filters[8].push_back("console");

    // check to see if any of the registered reporters has been selected
    for(auto& curr : getReporters()) {
        if(matchesAny(curr.first.second.c_str(), p->filters[8], false, p->case_sensitive))
            p->reporters_currently_used.push_back(curr.second(*g_cs));
    }

    // TODO: check if there is nothing in reporters_currently_used

    // prepend all listeners
    for(auto& curr : getListeners())
        p->reporters_currently_used.insert(p->reporters_currently_used.begin(), curr.second(*g_cs));

#ifdef DOCTEST_PLATFORM_WINDOWS
    if(isDebuggerActive() && p->no_debug_output == false)
        p->reporters_currently_used.push_back(new DebugOutputWindowReporter(*g_cs));
#endif // DOCTEST_PLATFORM_WINDOWS

    // handle version, help and no_run
    if(p->no_run || p->version || p->help || p->list_reporters) {
        DOCTEST_ITERATE_THROUGH_REPORTERS(report_query, QueryData());

        return cleanup_and_return();
    }

    std::vector<const TestCase*> testArray;
    for(auto& curr : getRegisteredTests())
        testArray.push_back(&curr);
    p->numTestCases = testArray.size();

    // sort the collected records
    if(!testArray.empty()) {
        if(p->order_by.compare("file", true) == 0) {
            std::sort(testArray.begin(), testArray.end(), fileOrderComparator);
        } else if(p->order_by.compare("suite", true) == 0) {
            std::sort(testArray.begin(), testArray.end(), suiteOrderComparator);
        } else if(p->order_by.compare("name", true) == 0) {
            std::sort(testArray.begin(), testArray.end(), nameOrderComparator);
        } else if(p->order_by.compare("rand", true) == 0) {
            std::srand(p->rand_seed);

            // random_shuffle implementation
            const auto first = &testArray[0];
            for(size_t i = testArray.size() - 1; i > 0; --i) {
                int idxToSwap = std::rand() % (i + 1);

                const auto temp = first[i];

                first[i]         = first[idxToSwap];
                first[idxToSwap] = temp;
            }
        } else if(p->order_by.compare("none", true) == 0) {
            // means no sorting - beneficial for death tests which call into the executable
            // with a specific test case in mind - we don't want to slow down the startup times
        }
    }

    std::set<String> testSuitesPassingFilt;

    bool                             query_mode = p->count || p->list_test_cases ||
p->list_test_suites;
    std::vector<const TestCaseData*> queryResults;

    if(!query_mode)
        DOCTEST_ITERATE_THROUGH_REPORTERS(test_run_start, DOCTEST_EMPTY);

    // invoke the registered functions if they match the filter criteria (or just count them)
    for(auto& curr : testArray) {
        const auto& tc = *curr;

        bool skip_me = false;
        if(tc.m_skip && !p->no_skip)
            skip_me = true;

        if(!matchesAny(tc.m_file.c_str(), p->filters[0], true, p->case_sensitive))
            skip_me = true;
        if(matchesAny(tc.m_file.c_str(), p->filters[1], false, p->case_sensitive))
            skip_me = true;
        if(!matchesAny(tc.m_test_suite, p->filters[2], true, p->case_sensitive))
            skip_me = true;
        if(matchesAny(tc.m_test_suite, p->filters[3], false, p->case_sensitive))
            skip_me = true;
        if(!matchesAny(tc.m_name, p->filters[4], true, p->case_sensitive))
            skip_me = true;
        if(matchesAny(tc.m_name, p->filters[5], false, p->case_sensitive))
            skip_me = true;

        if(!skip_me)
            p->numTestCasesPassingFilters++;

        // skip the test if it is not in the execution range
        if((p->last < p->numTestCasesPassingFilters && p->first <= p->last) ||
           (p->first > p->numTestCasesPassingFilters))
            skip_me = true;

        if(skip_me) {
            if(!query_mode)
                DOCTEST_ITERATE_THROUGH_REPORTERS(test_case_skipped, tc);
            continue;
        }

        // do not execute the test if we are to only count the number of filter passing tests
        if(p->count)
            continue;

        // print the name of the test and don't execute it
        if(p->list_test_cases) {
            queryResults.push_back(&tc);
            continue;
        }

        // print the name of the test suite if not done already and don't execute it
        if(p->list_test_suites) {
            if((testSuitesPassingFilt.count(tc.m_test_suite) == 0) && tc.m_test_suite[0] != '\0') {
                queryResults.push_back(&tc);
                testSuitesPassingFilt.insert(tc.m_test_suite);
                p->numTestSuitesPassingFilters++;
            }
            continue;
        }

        // execute the test if it passes all the filtering
        {
            p->currentTest = &tc;

            p->failure_flags = TestCaseFailureReason::None;
            p->seconds       = 0;

            // reset atomic counters
            p->numAssertsFailedCurrentTest_atomic = 0;
            p->numAssertsCurrentTest_atomic       = 0;

            p->fullyTraversedSubcases.clear();

            DOCTEST_ITERATE_THROUGH_REPORTERS(test_case_start, tc);

            p->timer.start();

            bool run_test = true;

            do {
                // reset some of the fields for subcases (except for the set of fully passed ones)
                p->reachedLeaf = false;
                // May not be empty if previous subcase exited via exception.
                p->subcaseStack.clear();
                p->currentSubcaseDepth = 0;

                p->shouldLogCurrentException = true;

                // reset stuff for logging with INFO()
                p->stringifiedContexts.clear();

#ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
```

```
                try {
#endif // DOCTEST_CONFIG_NO_EXCEPTIONS
// MSVC 2015 diagnoses fatalConditionHandler as unused (because reset() is a static method)
DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4101) // unreferenced local variable
                    FatalConditionHandler fatalConditionHandler; // Handle signals
                    // execute the test
                    tc.m_test();
                    fatalConditionHandler.reset();
DOCTEST_MSVC_SUPPRESS_WARNING_POP
#ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
                } catch(const TestFailureException&) {
                    p->failure_flags |= TestCaseFailureReason::AssertFailure;
                } catch(...) {
                    DOCTEST_ITERATE_THROUGH_REPORTERS(test_case_exception,
                                                      {translateActiveException(), false});
                    p->failure_flags |= TestCaseFailureReason::Exception;
                }
#endif // DOCTEST_CONFIG_NO_EXCEPTIONS

                // exit this loop if enough assertions have failed - even if there are more subcases
                if(p->abort_after > 0 &&
                   p->numAssertsFailed + p->numAssertsFailedCurrentTest_atomic >= p->abort_after) {
                    run_test = false;
                    p->failure_flags |= TestCaseFailureReason::TooManyFailedAsserts;
                }

                if(!p->nextSubcaseStack.empty() && run_test)
                    DOCTEST_ITERATE_THROUGH_REPORTERS(test_case_reenter, tc);
                if(p->nextSubcaseStack.empty())
                    run_test = false;
            } while(run_test);

            p->finalizeTestCaseData();

            DOCTEST_ITERATE_THROUGH_REPORTERS(test_case_end, *g_cs);

            p->currentTest = nullptr;

            // stop executing tests if enough assertions have failed
            if(p->abort_after > 0 && p->numAssertsFailed >= p->abort_after)
                break;
        }
    }

    if(!query_mode) {
        DOCTEST_ITERATE_THROUGH_REPORTERS(test_run_end, *g_cs);
    } else {
        QueryData qdata;
        qdata.run_stats = g_cs;
        qdata.data      = queryResults.data();
        qdata.num_data  = unsigned(queryResults.size());
        DOCTEST_ITERATE_THROUGH_REPORTERS(report_query, qdata);
    }

    return cleanup_and_return();
}

DOCTEST_DEFINE_INTERFACE(IReporter)

int IReporter::get_num_active_contexts() { return detail::g_infoContexts.size(); }
const IContextScope* const* IReporter::get_active_contexts() {
    return get_num_active_contexts() ? &detail::g_infoContexts[0] : nullptr;
}

int IReporter::get_num_stringified_contexts() { return detail::g_cs->stringifiedContexts.size(); }
const String* IReporter::get_stringified_contexts() {
    return get_num_stringified_contexts() ? &detail::g_cs->stringifiedContexts[0] : nullptr;
}

namespace detail {
    void registerReporterImpl(const char* name, int priority, reporterCreatorFunc c, bool isReporter)
{
        if(isReporter)
            getReporters().insert(reporterMap::value_type(reporterMap::key_type(priority, name), c));
        else
            getListeners().insert(reporterMap::value_type(reporterMap::key_type(priority, name), c));
    }
} // namespace detail

} // namespace doctest

#endif // DOCTEST_CONFIG_DISABLE

#ifdef DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4007) // 'function' : must be 'attribute' - see issue #182
int main(int argc, char** argv) { return doctest::Context(argc, argv).run(); }
DOCTEST_MSVC_SUPPRESS_WARNING_POP
```

```
#endif // DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN

DOCTEST_CLANG_SUPPRESS_WARNING_POP
DOCTEST_MSVC_SUPPRESS_WARNING_POP
DOCTEST_GCC_SUPPRESS_WARNING_POP

DOCTEST_SUPPRESS_COMMON_WARNINGS_POP

#endif // DOCTEST_LIBRARY_IMPLEMENTATION
#endif // DOCTEST_CONFIG_IMPLEMENT

#ifdef DOCTEST_UNDEF_WIN32_LEAN_AND_MEAN
#undef WIN32_LEAN_AND_MEAN
#undef DOCTEST_UNDEF_WIN32_LEAN_AND_MEAN
#endif // DOCTEST_UNDEF_WIN32_LEAN_AND_MEAN

#ifdef DOCTEST_UNDEF_NOMINMAX
#undef NOMINMAX
#undef DOCTEST_UNDEF_NOMINMAX
#endif // DOCTEST_UNDEF_NOMINMAX
```

## 6.4 vector.h

```
#ifndef VECTOR_H
#define VECTOR_H

#include <cstddef>
#include <limits>
#include <utility>
#include <algorithm>
#include <stdexcept>
#include <initializer_list>
#include <iostream>

template <typename T>
class Vector {
private:
    T* vec_;
    size_t capacity_;
    size_t size_;
    size_t reallocations; // Skaitiklis perskirstymams

    void reallocate(size_t new_capacity) {
        T* new_vec = new T[new_capacity];
        try {
            std::move(vec_, vec_ + size_, new_vec);
        } catch (...) {
            delete[] new_vec;
            throw;
        }
        delete[] vec_;
        vec_ = new_vec;
        capacity_ = new_capacity;
        reallocations++;
    }

public:
    // Member types
    using value_type = T;
    using size_type = size_t;
    using reference = T&;
    using const_reference = const T&;
    using iterator = T*;
    using const_iterator = const T*;
    using reverse_iterator = std::reverse_iterator<iterator>;
    using const_reverse_iterator = std::reverse_iterator<const_iterator>;

    // Constructors
    Vector() : vec_(nullptr), capacity_(0), size_(0), reallocations(0) {}

    explicit Vector(size_type count)
        : vec_(new T[count]), capacity_(count), size_(count), reallocations(0) {
        std::fill_n(vec_, size_, T());
    }

    Vector(size_type count, const T& value)
        : vec_(new T[count]), capacity_(count), size_(count), reallocations(0) {
        std::fill_n(vec_, size_, value);
    }

    Vector(std::initializer_list<T> list)
        : vec_(new T[list.size()]), capacity_(list.size()), size_(list.size()), reallocations(0) {
        std::copy(list.begin(), list.end(), vec_);
    }

    // Copy constructor
    Vector(const Vector& other)
        : vec_(new T[other.capacity_]), capacity_(other.capacity_), size_(other.size_),
reallocations(0) {
        std::copy(other.vec_, other.vec_ + size_, vec_);
    }

    // Move constructor
    Vector(Vector&& other) noexcept
        : vec_(other.vec_), capacity_(other.capacity_), size_(other.size_),
reallocations(other.reallocations) {
        other.vec_ = nullptr;
        other.capacity_ = 0;
        other.size_ = 0;
        other.reallocations = 0;
    }

    // Destructor
    ~Vector() {
        delete[] vec_;
    }

    // Assignment operators
    Vector& operator=(const Vector& other) {
        if (this != &other) {
            Vector temp(other);
            swap(temp);
        }
        return *this;
    }

    Vector& operator=(Vector&& other) noexcept {
        if (this != &other) {
            delete[] vec_;
            vec_ = other.vec_;
            size_ = other.size_;
            capacity_ = other.capacity_;
            reallocations = other.reallocations;
            other.vec_ = nullptr;
            other.size_ = 0;
            other.capacity_ = 0;
            other.reallocations = 0;
        }
        return *this;
    }

    // Element access (NEKEISTA)
    reference operator[](size_type pos) { return vec_[pos]; }
    const_reference operator[](size_type pos) const { return vec_[pos]; }
    reference at(size_type pos) {
        if (pos >= size_) throw std::out_of_range("out of range");
        return vec_[pos];
    }
    const_reference at(size_type pos) const {
        if (pos >= size_) throw std::out_of_range("out of range");
        return vec_[pos];
    }
    reference front() { return vec_[0]; }
    const_reference front() const { return vec_[0]; }
    reference back() { return vec_[size_ - 1]; }
    const_reference back() const { return vec_[size_ - 1]; }
    T* data() noexcept { return vec_; }
    const T* data() const noexcept { return vec_; }

    // Iterators (NEKEISTA)
    iterator begin() noexcept { return vec_; }
    const_iterator begin() const noexcept { return vec_; }
    const_iterator cbegin() const noexcept { return vec_; }
    iterator end() noexcept { return vec_ + size_; }
    const_iterator end() const noexcept { return vec_ + size_; }
    const_iterator cend() const noexcept { return vec_ + size_; }
    reverse_iterator rbegin() noexcept { return reverse_iterator(end()); }
    const_reverse_iterator rbegin() const noexcept { return const_reverse_iterator(end()); }
    const_reverse_iterator crbegin() const noexcept { return const_reverse_iterator(end()); }
    reverse_iterator rend() noexcept { return reverse_iterator(begin()); }
    const_reverse_iterator rend() const noexcept { return const_reverse_iterator(begin()); }
    const_reverse_iterator crend() const noexcept { return const_reverse_iterator(begin()); }

    // Capacity
    bool empty() const noexcept { return size_ == 0; }
    size_type size() const noexcept { return size_; }
    size_type capacity() const noexcept { return capacity_; }
    size_type getReallocations() const { return reallocations; }
    size_type max_size() const noexcept { return std::numeric_limits<size_type>::max() / sizeof(T); }

```

```cpp
    void reserve(size_type new_cap) {
        if (new_cap > capacity_) {
            reallocate(new_cap);
        }
    }

    void shrink_to_fit() {
        if (size_ < capacity_) {
            reallocate(size_);
        }
    }

    // Modifiers
    void clear() noexcept { size_ = 0; }

    iterator insert(const_iterator pos, const T& value) {
        return emplace(pos, value);
    }

    iterator insert(const_iterator pos, T&& value) {
        return emplace(pos, std::move(value));
    }

    iterator erase(const_iterator pos) {
        if (pos < cbegin() || pos >= cend()) throw std::out_of_range("out of range");
        iterator non_const_pos = begin() + (pos - cbegin());
        std::move(non_const_pos + 1, end(), non_const_pos);
        --size_;
        return non_const_pos;
    }

    iterator erase(const_iterator first, const_iterator last) {
        if (first < cbegin() || last > cend() || first > last) throw std::out_of_range("out of range");
        iterator non_const_first = begin() + (first - cbegin());
        iterator non_const_last = begin() + (last - cbegin());
        std::move(non_const_last, end(), non_const_first);
        size_ -= (last - first);
        return non_const_first;
    }

    void push_back(const T& value) {
        if (size_ == capacity_) reserve(capacity_ == 0 ? 1 : capacity_ * 2);
        vec_[size_++] = value;
    }

    void push_back(T&& value) {
        if (size_ == capacity_) reserve(capacity_ == 0 ? 1 : capacity_ * 2);
        vec_[size_++] = std::move(value);
    }

    template <typename... Args>
    reference emplace_back(Args&&... args) {
        if (size_ >= capacity_) reserve(capacity_ == 0 ? 1 : capacity_ * 2);
        new (&vec_[size_]) T(std::forward<Args>(args)...);
        return vec_[size_++];
    }

    template <typename... Args>
    iterator emplace(const_iterator pos, Args&&... args) {
        if (pos < cbegin() || pos > cend()) throw std::out_of_range("Vector::emplace - iterator out of range");
        size_type offset = pos - cbegin();
        if (size_ >= capacity_) reserve(capacity_ == 0 ? 1 : capacity_ * 2);
        iterator insert_pos = begin() + offset;
        if (insert_pos != end()) {
            std::move_backward(insert_pos, end(), end() + 1);
        }
        new (&(*insert_pos)) T(std::forward<Args>(args)...);
        ++size_;
        return insert_pos;
    }

    void pop_back() {
        if (size_ > 0) --size_;
    }

    void resize(size_type count) {
        if (count > capacity_) reserve(count);
        if (count > size_) std::fill(vec_ + size_, vec_ + count, T());
        size_ = count;
    }

    void resize(size_type count, const value_type& value) {
        if (count > capacity_) reserve(count);
        if (count > size_) std::fill(vec_ + size_, vec_ + count, value);
        size_ = count;
    }

    void swap(Vector& other) noexcept {
        std::swap(vec_, other.vec_);
        std::swap(size_, other.size_);
        std::swap(capacity_, other.capacity_);
        std::swap(reallocations, other.reallocations);
    }

    // Comparison operators (NEKEISTA)
    bool operator==(const Vector& other) const {
        if (size_ != other.size_) return false;
        return std::equal(begin(), end(), other.begin());
    }

    bool operator!=(const Vector& other) const { return !(*this == other); }
    bool operator<(const Vector& other) const {
        return std::lexicographical_compare(begin(), end(), other.begin(), other.end());
    }
    bool operator<=(const Vector& other) const { return !(*this > other); }
    bool operator>(const Vector& other) const { return other < *this; }
    bool operator>=(const Vector& other) const { return !(*this < other); }
};

// Non-member swap function
template <typename T>
void swap(Vector<T>& lhs, Vector<T>& rhs) noexcept {
    lhs.swap(rhs);
}

#endif // VECTOR_H
```

## 6.5 vector.h

```
#ifndef VECTOR_H
#define VECTOR_H

#include <cstddef>
#include <limits>
#include <utility>
#include <algorithm>
#include <stdexcept>
#include <initializer_list>
#include <iostream>

template <typename T>
class Vector {
private:
    T* vec_;
    size_t capacity_;
    size_t size_;
    size_t reallocations; // Skaitiklis perskirstymams

    void reallocate(size_t new_capacity) {
        T* new_vec = new T[new_capacity];
        try {
            std::move(vec_, vec_ + size_, new_vec);
        } catch (...) {
            delete[] new_vec;
            throw;
        }
        delete[] vec_;
        vec_ = new_vec;
        capacity_ = new_capacity;
        reallocations++;
    }

public:
    // Member types
    using value_type = T;
    using size_type = size_t;
    using reference = T&;
    using const_reference = const T&;
    using iterator = T*;
    using const_iterator = const T*;
    using reverse_iterator = std::reverse_iterator<iterator>;
    using const_reverse_iterator = std::reverse_iterator<const_iterator>;

    // Constructors
    Vector() : vec_(nullptr), capacity_(0), size_(0), reallocations(0) {}

    explicit Vector(size_type count)
        : vec_(new T[count]), capacity_(count), size_(count), reallocations(0) {
```

```cpp
        std::fill_n(vec_, size_, T());
    }

    Vector(size_type count, const T& value)
        : vec_(new T[count]), capacity_(count), size_(count), reallocations(0) {
        std::fill_n(vec_, size_, value);
    }

    Vector(std::initializer_list<T> list)
        : vec_(new T[list.size()]), capacity_(list.size()), size_(list.size()), reallocations(0) {
        std::copy(list.begin(), list.end(), vec_);
    }

    // Copy constructor
    Vector(const Vector& other)
        : vec_(new T[other.capacity_]), capacity_(other.capacity_), size_(other.size_),
reallocations(0) {
        std::copy(other.vec_, other.vec_ + size_, vec_);
    }

    // Move constructor
    Vector(Vector&& other) noexcept
        : vec_(other.vec_), capacity_(other.capacity_), size_(other.size_),
reallocations(other.reallocations) {
        other.vec_ = nullptr;
        other.capacity_ = 0;
        other.size_ = 0;
        other.reallocations = 0;
    }

    // Destructor
    ~Vector() {
        delete[] vec_;
    }

    // Assignment operators
    Vector& operator=(const Vector& other) {
        if (this != &other) {
            Vector temp(other);
            swap(temp);
        }
        return *this;
    }

    Vector& operator=(Vector&& other) noexcept {
        if (this != &other) {
            delete[] vec_;
            vec_ = other.vec_;
            size_ = other.size_;
            capacity_ = other.capacity_;
            reallocations = other.reallocations;
            other.vec_ = nullptr;
            other.size_ = 0;
            other.capacity_ = 0;
            other.reallocations = 0;
        }
        return *this;
    }

    // Element access (NEKEISTA)
    reference operator[](size_type pos) { return vec_[pos]; }
    const_reference operator[](size_type pos) const { return vec_[pos]; }
    reference at(size_type pos) {
        if (pos >= size_) throw std::out_of_range("out of range");
        return vec_[pos];
    }
    const_reference at(size_type pos) const {
        if (pos >= size_) throw std::out_of_range("out of range");
        return vec_[pos];
    }
    reference front() { return vec_[0]; }
    const_reference front() const { return vec_[0]; }
    reference back() { return vec_[size_ - 1]; }
    const_reference back() const { return vec_[size_ - 1]; }
    T* data() noexcept { return vec_; }
    const T* data() const noexcept { return vec_; }

    // Iterators (NEKEISTA)
    iterator begin() noexcept { return vec_; }
    const_iterator begin() const noexcept { return vec_; }
    const_iterator cbegin() const noexcept { return vec_; }
    iterator end() noexcept { return vec_ + size_; }
    const_iterator end() const noexcept { return vec_ + size_; }
    const_iterator cend() const noexcept { return vec_ + size_; }
    reverse_iterator rbegin() noexcept { return reverse_iterator(end()); }
    const_reverse_iterator rbegin() const noexcept { return const_reverse_iterator(end()); }
    const_reverse_iterator crbegin() const noexcept { return const_reverse_iterator(end()); }
    reverse_iterator rend() noexcept { return reverse_iterator(begin()); }
    const_reverse_iterator rend() const noexcept { return const_reverse_iterator(begin()); }
    const_reverse_iterator crend() const noexcept { return const_reverse_iterator(begin()); }

    // Capacity
    bool empty() const noexcept { return size_ == 0; }
    size_type size() const noexcept { return size_; }
    size_type capacity() const noexcept { return capacity_; }
    size_type getReallocations() const { return reallocations; }
    size_type max_size() const noexcept { return std::numeric_limits<size_type>::max() / sizeof(T); }

    void reserve(size_type new_cap) {
        if (new_cap > capacity_) {
            reallocate(new_cap);
        }
    }

    void shrink_to_fit() {
        if (size_ < capacity_) {
            reallocate(size_);
        }
    }

    // Modifiers
    void clear() noexcept { size_ = 0; }

    iterator insert(const_iterator pos, const T& value) {
        return emplace(pos, value);
    }

    iterator insert(const_iterator pos, T&& value) {
        return emplace(pos, std::move(value));
    }

    iterator erase(const_iterator pos) {
        if (pos < cbegin() || pos >= cend()) throw std::out_of_range("out of range");
        iterator non_const_pos = begin() + (pos - cbegin());
        std::move(non_const_pos + 1, end(), non_const_pos);
        --size_;
        return non_const_pos;
    }

    iterator erase(const_iterator first, const_iterator last) {
        if (first < cbegin() || last > cend() || first > last) throw std::out_of_range("out of range");
        iterator non_const_first = begin() + (first - cbegin());
        iterator non_const_last = begin() + (last - cbegin());
        std::move(non_const_last, end(), non_const_first);
        size_ -= (last - first);
        return non_const_first;
    }

    void push_back(const T& value) {
        if (size_ == capacity_) reserve(capacity_ == 0 ? 1 : capacity_ * 2);
        vec_[size_++] = value;
    }

    void push_back(T&& value) {
        if (size_ == capacity_) reserve(capacity_ == 0 ? 1 : capacity_ * 2);
        vec_[size_++] = std::move(value);
    }

    template <typename... Args>
    reference emplace_back(Args&&... args) {
        if (size_ >= capacity_) reserve(capacity_ == 0 ? 1 : capacity_ * 2);
        new (&vec_[size_]) T(std::forward<Args>(args)...);
        return vec_[size_++];
    }

    template <typename... Args>
    iterator emplace(const_iterator pos, Args&&... args) {
        if (pos < cbegin() || pos > cend()) throw std::out_of_range("Vector::emplace - iterator out of range");
        size_type offset = pos - cbegin();
        if (size_ >= capacity_) reserve(capacity_ == 0 ? 1 : capacity_ * 2);
        iterator insert_pos = begin() + offset;
        if (insert_pos != end()) {
            std::move_backward(insert_pos, end(), end() + 1);
        }
        new (&(*insert_pos)) T(std::forward<Args>(args)...);
        ++size_;
        return insert_pos;
    }

    void pop_back() {
        if (size_ > 0) --size_;
    }

    void resize(size_type count) {
        if (count > capacity_) reserve(count);
        if (count > size_) std::fill(vec_ + size_, vec_ + count, T());
        size_ = count;
    }

    void resize(size_type count, const value_type& value) {
        if (count > capacity_) reserve(count);
        if (count > size_) std::fill(vec_ + size_, vec_ + count, value);
        size_ = count;
    }

    void swap(Vector& other) noexcept {
        std::swap(vec_, other.vec_);
        std::swap(size_, other.size_);
        std::swap(capacity_, other.capacity_);
        std::swap(reallocations, other.reallocations);
    }

    // Comparison operators (NEKEISTA)
    bool operator==(const Vector& other) const {
        if (size_ != other.size_) return false;
        return std::equal(begin(), end(), other.begin());
    }

    bool operator!=(const Vector& other) const { return !(*this == other); }
    bool operator<(const Vector& other) const {
        return std::lexicographical_compare(begin(), end(), other.begin(), other.end());
    }
    bool operator<=(const Vector& other) const { return !(*this > other); }
    bool operator>(const Vector& other) const { return other < *this; }
    bool operator>=(const Vector& other) const { return !(*this < other); }
};

// Non-member swap function
template <typename T>
void swap(Vector<T>& lhs, Vector<T>& rhs) noexcept {
    lhs.swap(rhs);
}

#endif // VECTOR_H
```

## 6.6 zmogus.h

```
#ifndef ZMOGUS_H
#define ZMOGUS_H

#include <string>

class Zmogus {
protected:
    std::string vardas_;
    std::string pavarde_;

public:
    Zmogus() = default;
    Zmogus(const std::string& vardas, const std::string& pavarde)
        : vardas_(vardas), pavarde_(pavarde) {}

    virtual ~Zmogus() = default;

    std::string vardas() const { return vardas_; }
    std::string pavarde() const { return pavarde_; }

    void setVardas(const std::string& vardas) { vardas_ = vardas; }
    void setPavarde(const std::string& pavarde) { pavarde_ = pavarde; }

    virtual std::ostream& spausdinti(std::ostream& os) const = 0;
};

#endif
```

# Index