

Vector klase

Generated by Doxygen 1.13.2



<b>1 3.0 nuosavos Vector klasės testavimas</b>	<b>1</b>
1.1 Aprašymas	1
1.2 5 funkcijų pavyzdžiai	1
1.2.1 1. operator[]	1
1.2.2 2. push_back	1
1.2.3 3. at()	1
1.2.4 4. operator==	2
1.2.5 5. empty()	2
1.3 Testavimas	2
1.4 Efektyvumo analizė	2
1.5 Perskirstymų skaičius	2
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Class Documentation</b>	<b>9</b>
5.1 doctest::Approx Struct Reference	9
5.2 doctest::AssertData Struct Reference	10
5.3 std::basic_istream< charT, traits > Class Template Reference	10
5.4 std::basic_ostream< charT, traits > Class Template Reference	11
5.5 std::char_traits< charT > Struct Template Reference	11
5.6 doctest::Contains Class Reference	11
5.7 doctest::Context Class Reference	11
5.8 doctest::ContextOptions Struct Reference	12
5.8.1 Detailed Description	12
5.9 doctest::detail::ContextScope< L > Class Template Reference	13
5.9.1 Member Function Documentation	13
5.9.1.1 stringify()	13
5.10 doctest::detail::ContextScopeBase Struct Reference	14
5.11 doctest::CurrentTestCaseStats Struct Reference	14
5.12 doctest::detail::deferred_false< T > Struct Template Reference	15
5.13 doctest::detail::types::enable_if< COND, T > Struct Template Reference	15
5.14 doctest::detail::types::enable_if< true, T > Struct Template Reference	15
5.15 doctest::detail::ExceptionTranslator< T > Class Template Reference	16
5.15.1 Detailed Description	16
5.15.2 Member Function Documentation	16
5.15.2.1 translate()	16
5.16 doctest::detail::Expression_lhs< L > Struct Template Reference	16

5.16.1 Member Function Documentation	17
5.16.1.1 operator Result()	17
5.17 doctest::detail::ExpressionComposer Struct Reference	17
5.18 doctest::detail::types::false_type Struct Reference	17
5.19 doctest::detail::filldata< T > Struct Template Reference	18
5.20 doctest::detail::filldata< const char[N]> Struct Template Reference	18
5.21 doctest::detail::filldata< const void * > Struct Reference	18
5.22 doctest::detail::filldata< T * > Struct Template Reference	19
5.23 doctest::detail::filldata< T[N]> Struct Template Reference	19
5.24 doctest::detail::has_insertion_operator< T, typename > Struct Template Reference	19
5.25 doctest::detail::has_insertion_operator< T, decltype(operator<<(declval< std::ostream & >()), declval< const T & >()), void())> Struct Template Reference	20
5.26 doctest::IContextScope Struct Reference	20
5.27 doctest::detail::IExceptionTranslator Struct Reference	21
5.28 doctest::IReporter Struct Reference	21
5.29 doctest::detail::types::is_array< T > Struct Template Reference	22
5.30 doctest::detail::types::is_array< T[SIZE]> Struct Template Reference	22
5.31 doctest::detail::types::is_enum< T > Struct Template Reference	23
5.32 doctest::detail::types::is_pointer< T > Struct Template Reference	23
5.33 doctest::detail::types::is_pointer< T * > Struct Template Reference	23
5.34 doctest::detail::types::is_rvalue_reference< T > Struct Template Reference	24
5.35 doctest::detail::types::is_rvalue_reference< T && > Struct Template Reference	24
5.36 doctest::IsNaN< F > Struct Template Reference	25
5.37 doctest::detail::MessageBuilder Struct Reference	25
5.38 doctest::MessageData Struct Reference	26
5.39 doctest::QueryData Struct Reference	26
5.40 doctest::detail::RelationalComparator< int, L, R > Struct Template Reference	27
5.41 doctest::detail::types::remove_const< T > Struct Template Reference	27
5.42 doctest::detail::types::remove_const< const T > Struct Template Reference	27
5.43 doctest::detail::types::remove_reference< T > Struct Template Reference	27
5.44 doctest::detail::types::remove_reference< T & > Struct Template Reference	28
5.45 doctest::detail::types::remove_reference< T && > Struct Template Reference	28
5.46 doctest::detail::Result Struct Reference	28
5.47 doctest::detail::ResultBuilder Struct Reference	29
5.47.1 Member Function Documentation	30
5.47.1.1 unary_assert()	30
5.48 doctest::detail::should_stringify_as_underlying_type< T > Struct Template Reference	30
5.49 doctest::String Class Reference	30
5.50 doctest::AssertData::StringContains Class Reference	31
5.51 doctest::StringMaker< T > Struct Template Reference	31
5.52 doctest::detail::StringMakerBase< C > Struct Template Reference	32
5.53 doctest::detail::StringMakerBase< true > Struct Reference	32

5.54 Studentas Class Reference . . . . .	32
5.54.1 Member Function Documentation . . . . .	33
5.54.1.1 spausdinti() . . . . .	33
5.55 doctest::detail::Subcase Struct Reference . . . . .	33
5.56 doctest::SubcaseSignature Struct Reference . . . . .	34
5.57 doctest::detail::TestCase Struct Reference . . . . .	34
5.58 doctest::TestCaseData Struct Reference . . . . .	35
5.59 doctest::TestCaseException Struct Reference . . . . .	36
5.60 doctest::detail::TestFailureException Struct Reference . . . . .	36
5.61 doctest::TestRunStats Struct Reference . . . . .	36
5.62 doctest::detail::TestSuite Struct Reference . . . . .	36
5.63 doctest::detail::types::true_type Struct Reference . . . . .	37
5.64 std::tuple< Types > Class Template Reference . . . . .	37
5.65 doctest::detail::types::underlying_type< T > Struct Template Reference . . . . .	37
5.66 Vector< T > Class Template Reference . . . . .	38
5.67 Zmogus Class Reference . . . . .	40
<b>6 File Documentation</b>	<b>43</b>
6.1 funkcijos.h . . . . .	43
6.2 studentas.h . . . . .	43
6.3 doctest.h . . . . .	44
6.4 vector.h . . . . .	129
6.5 vector.h . . . . .	132
6.6 zmogus.h . . . . .	135
<b>Index</b>	<b>137</b>



# 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 dinامينius 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:

doctest::Approx	9
doctest::AssertData	10
doctest::detail::ResultBuilder	29
std::basic_istream< charT, traits >	10
std::basic_ostream< charT, traits >	11
std::char_traits< charT >	11
doctest::Contains	11
doctest::Context	11
doctest::ContextOptions	12
doctest::CurrentTestCaseStats	14
doctest::detail::types::enable_if< COND, T >	15
doctest::detail::types::enable_if< true, T >	15
doctest::detail::Expression_lhs< L >	16
doctest::detail::ExpressionDecomposer	17
doctest::detail::types::false_type	17
doctest::detail::has_insertion_operator< T, decltype(operator<<(declval< std::ostream & >()), declval< const T & >()), void())>	20
doctest::detail::types::is_array< T[SIZE]>	22
doctest::detail::types::is_pointer< T * >	23
doctest::detail::types::is_rvalue_reference< T && >	24
doctest::detail::deferred_false< T >	15
doctest::detail::has_insertion_operator< T, typename >	19
doctest::detail::types::is_array< T >	22
doctest::detail::types::is_pointer< T >	23
doctest::detail::types::is_rvalue_reference< T >	24
doctest::detail::filldata< T >	18
doctest::detail::filldata< const char[N]>	18
doctest::detail::filldata< const void * >	18
doctest::detail::filldata< T * >	19
doctest::detail::filldata< T[N]>	19
doctest::IContextScope	20
doctest::detail::ContextScopeBase	14
doctest::detail::ContextScope< L >	13
doctest::detail::IExceptionTranslator	21

doctest::detail::ExceptionTranslator< T > . . . . .	16
doctest::IReporter . . . . .	21
doctest::detail::types::is_enum< T > . . . . .	23
doctest::IsNaN< F > . . . . .	25
doctest::MessageData . . . . .	26
doctest::detail::MessageBuilder . . . . .	25
doctest::QueryData . . . . .	26
doctest::detail::RelationalComparator< int, L, R > . . . . .	27
doctest::detail::types::remove_const< T > . . . . .	27
doctest::detail::types::remove_const< const T > . . . . .	27
doctest::detail::types::remove_reference< T > . . . . .	27
doctest::detail::types::remove_reference< T & > . . . . .	28
doctest::detail::types::remove_reference< T && > . . . . .	28
doctest::detail::Result . . . . .	28
doctest::detail::should_stringify_as_underlying_type< T > . . . . .	30
doctest::String . . . . .	30
doctest::AssertData::StringContains . . . . .	31
doctest::detail::StringMakerBase< C > . . . . .	32
doctest::detail::StringMakerBase< detail::has_insertion_operator< T >::value  detail::types::is_pointer< T >::value  detail::types::is_array< T >::value > . . . . .	32
doctest::StringMaker< T > . . . . .	31
doctest::detail::StringMakerBase< true > . . . . .	32
doctest::detail::Subcase . . . . .	33
doctest::SubcaseSignature . . . . .	34
doctest::TestCaseData . . . . .	35
doctest::detail::TestCase . . . . .	34
doctest::TestCaseException . . . . .	36
doctest::detail::TestFailureException . . . . .	36
doctest::TestRunStats . . . . .	36
doctest::detail::TestSuite . . . . .	36
doctest::detail::types::true_type . . . . .	37
doctest::detail::has_insertion_operator< T, decltype(operator<<(declval< std::ostream & >(), declval< const T & >()), void())> . . . . .	20
doctest::detail::types::is_array< T[SIZE]> . . . . .	22
doctest::detail::types::is_pointer< T * > . . . . .	23
doctest::detail::types::is_rvalue_reference< T && > . . . . .	24
std::tuple< Types > . . . . .	37
doctest::detail::types::underlying_type< T > . . . . .	37
Vector< T > . . . . .	38
Zmogus . . . . .	40
Studentas . . . . .	32

## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">doctest::Approx</a>	9
<a href="#">doctest::AssertData</a>	10
<a href="#">std::basic_istream&lt; charT, traits &gt;</a>	10
<a href="#">std::basic_ostream&lt; charT, traits &gt;</a>	11
<a href="#">std::char_traits&lt; charT &gt;</a>	11
<a href="#">doctest::Contains</a>	11
<a href="#">doctest::Context</a>	11
<a href="#">doctest::ContextOptions</a>	
OCLINT too many fields	12
<a href="#">doctest::detail::ContextScope&lt; L &gt;</a>	13
<a href="#">doctest::detail::ContextScopeBase</a>	14
<a href="#">doctest::CurrentTestCaseStats</a>	14
<a href="#">doctest::detail::deferred_false&lt; T &gt;</a>	15
<a href="#">doctest::detail::types::enable_if&lt; COND, T &gt;</a>	15
<a href="#">doctest::detail::types::enable_if&lt; true, T &gt;</a>	15
<a href="#">doctest::detail::ExceptionTranslator&lt; T &gt;</a>	
OCLINT destructor of virtual class	16
<a href="#">doctest::detail::Expression_lhs&lt; L &gt;</a>	16
<a href="#">doctest::detail::ExpressionDecomposer</a>	17
<a href="#">doctest::detail::types::false_type</a>	17
<a href="#">doctest::detail::filldata&lt; T &gt;</a>	18
<a href="#">doctest::detail::filldata&lt; const char[N]&gt;</a>	18
<a href="#">doctest::detail::filldata&lt; const void * &gt;</a>	18
<a href="#">doctest::detail::filldata&lt; T * &gt;</a>	19
<a href="#">doctest::detail::filldata&lt; T[N]&gt;</a>	19
<a href="#">doctest::detail::has_insertion_operator&lt; T, typename &gt;</a>	19
<a href="#">doctest::detail::has_insertion_operator&lt; T, decltype(operator&lt;&lt;(declval&lt; std::ostream &amp;&gt;(), declval&lt; const T &amp;&gt;()), void())&gt;</a>	
20	
<a href="#">doctest::!ContextScope</a>	20
<a href="#">doctest::detail::!ExceptionTranslator</a>	21
<a href="#">doctest::!Reporter</a>	21
<a href="#">doctest::detail::types::is_array&lt; T &gt;</a>	22
<a href="#">doctest::detail::types::is_array&lt; T[SIZE]&gt;</a>	22
<a href="#">doctest::detail::types::is_enum&lt; T &gt;</a>	23
<a href="#">doctest::detail::types::is_pointer&lt; T &gt;</a>	23

<a href="#">doctest::detail::types::is_pointer&lt; T * &gt;</a>	23
<a href="#">doctest::detail::types::is_rvalue_reference&lt; T &gt;</a>	24
<a href="#">doctest::detail::types::is_rvalue_reference&lt; T &amp;&amp; &gt;</a>	24
<a href="#">doctest::IsNaN&lt; F &gt;</a>	25
<a href="#">doctest::detail::MessageBuilder</a>	25
<a href="#">doctest::MessageData</a>	26
<a href="#">doctest::QueryData</a>	26
<a href="#">doctest::detail::RelationalComparator&lt; int, L, R &gt;</a>	27
<a href="#">doctest::detail::types::remove_const&lt; T &gt;</a>	27
<a href="#">doctest::detail::types::remove_const&lt; const T &gt;</a>	27
<a href="#">doctest::detail::types::remove_reference&lt; T &gt;</a>	27
<a href="#">doctest::detail::types::remove_reference&lt; T &amp; &gt;</a>	28
<a href="#">doctest::detail::types::remove_reference&lt; T &amp;&amp; &gt;</a>	28
<a href="#">doctest::detail::Result</a>	28
<a href="#">doctest::detail::ResultBuilder</a>	29
<a href="#">doctest::detail::should_stringify_as_underlying_type&lt; T &gt;</a>	30
<a href="#">doctest::String</a>	30
<a href="#">doctest::AssertData::StringContains</a>	31
<a href="#">doctest::StringMaker&lt; T &gt;</a>	31
<a href="#">doctest::detail::StringMakerBase&lt; C &gt;</a>	32
<a href="#">doctest::detail::StringMakerBase&lt; true &gt;</a>	32
<a href="#">Studentas</a>	32
<a href="#">doctest::detail::Subcase</a>	33
<a href="#">doctest::SubcaseSignature</a>	34
<a href="#">doctest::detail::TestCase</a>	34
<a href="#">doctest::TestCaseData</a>	35
<a href="#">doctest::TestCaseException</a>	36
<a href="#">doctest::detail::TestFailureException</a>	36
<a href="#">doctest::TestRunStats</a>	36
<a href="#">doctest::detail::TestSuite</a>	36
<a href="#">doctest::detail::types::true_type</a>	37
<a href="#">std::tuple&lt; Types &gt;</a>	37
<a href="#">doctest::detail::types::underlying_type&lt; T &gt;</a>	37
<a href="#">Vector&lt; T &gt;</a>	38
<a href="#">Zmogus</a>	40

# Chapter 4

## File Index

### 4.1 File List

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

<a href="#">funkcijos.h</a> . . . . .	43
<a href="#">studentas.h</a> . . . . .	43
<a href="#">vector.h</a> . . . . .	132
<a href="#">zmogus.h</a> . . . . .	135
Testavimas/ <a href="#">doctest.h</a> . . . . .	44
Testavimas/ <a href="#">vector.h</a> . . . . .	129



# 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

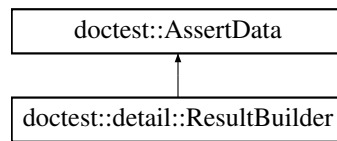
- DOctest\_INTERFACE friend bool **operator==** (double lhs, const [Approx](#) &rhs)
- DOctest\_INTERFACE friend bool **operator==** (const [Approx](#) &lhs, double rhs)
- DOctest\_INTERFACE friend bool **operator!=** (double lhs, const [Approx](#) &rhs)
- DOctest\_INTERFACE friend bool **operator!=** (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

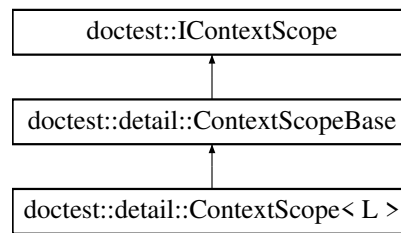
OCLINT too many fields.

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 >:



### 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]
  
```

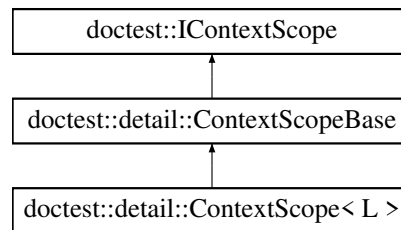
Implements [doctest::IContextScope](#).

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:



### 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

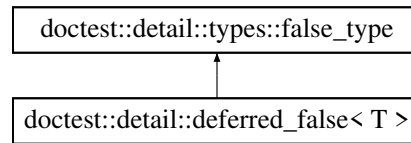
- int **numAssertsCurrentTest**
- int **numAssertsFailedCurrentTest**
- double **seconds**
- int **failure\_flags**
- bool **testCaseSuccess**

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 >:



### 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:

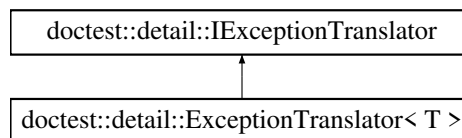
- Testavimas/doctest.h

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

OCLINT destructor of virtual class.

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

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



### 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

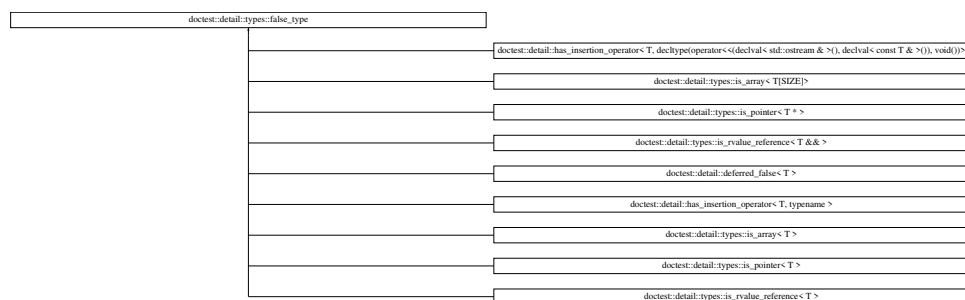
- assertType::Enum m\_at

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:



### 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

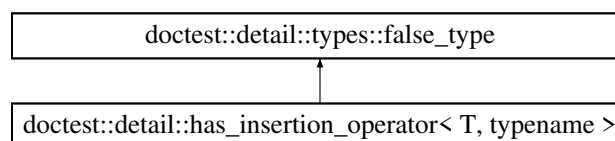
- static void **fill** ([std::ostream](#) \*stream, const T(&in)[N])
- static void **fill** ([std::ostream](#) \*stream, const T &in)

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 >:



### Additional Inherited Members

### Static Public Attributes inherited from [doctest::detail::types::false\\_type](#)

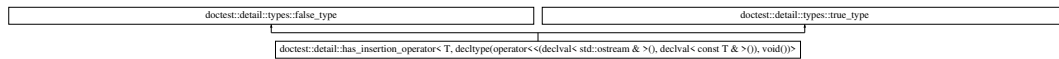
- static DOCTEST\_CONSTEXPR bool **value** = false

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())>:



### 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](#)

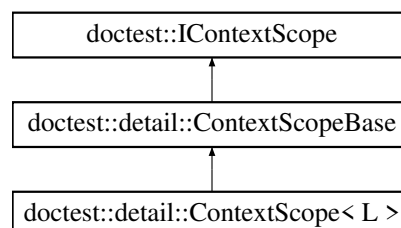
- static DOCTEST\_CONSTEXPR bool **value** = true

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:



### Public Member Functions

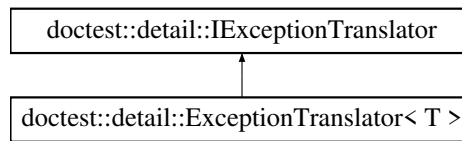
- virtual void **stringify** ([std::ostream](#) \*) const =0

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:



### 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

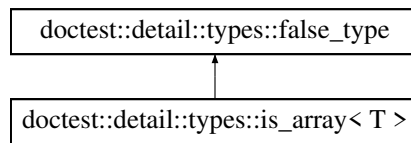
- static int **get\_num\_active\_contexts** ()
- static const [IContextScope](#) \*const \* **get\_active\_contexts** ()
- static int **get\_num\_stringified\_contexts** ()
- static const [String](#) \* **get\_stringified\_contexts** ()

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 >:



### Additional Inherited Members

#### Static Public Attributes inherited from [doctest::detail::types::false\\_type](#)

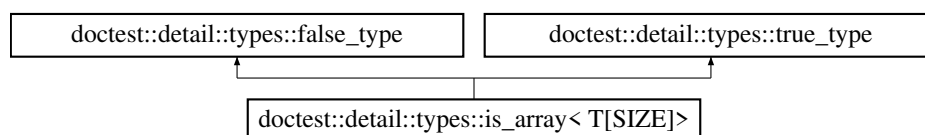
- static DOCTEST\_CONSTEXPR bool **value** = false

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]>:



### 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

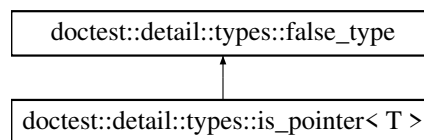
- static DOCTEST\_CONSTEXPR bool **value** = \_\_is\_enum(T)

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 >:



### Additional Inherited Members

#### Static Public Attributes inherited from [doctest::detail::types::false\\_type](#)

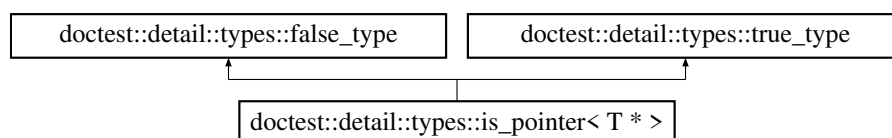
- static DOCTEST\_CONSTEXPR bool **value** = false

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 \* >:



### 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](#)

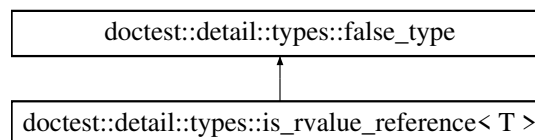
- static DOCTEST\_CONSTEXPR bool **value** = true

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 >:



### Additional Inherited Members

### Static Public Attributes inherited from [doctest::detail::types::false\\_type](#)

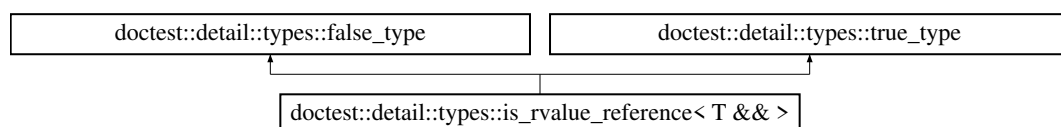
- static DOCTEST\_CONSTEXPR bool **value** = false

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 && >:



### 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

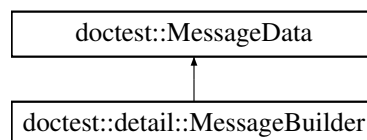
- F **value**
- bool **flipped**

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`**

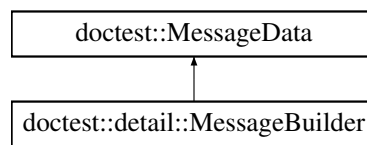
- `String m_string`
- `const char * m_file`
- `int m_line`
- `assertType::Enum m_severity`

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

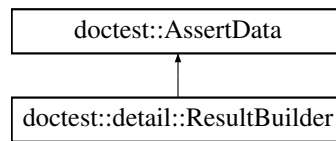
- bool **m\_passed**
- [String](#) **m\_decomp**

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 [StringContains](#) &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

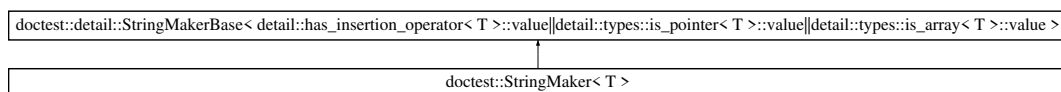
- **StringContains** (const [String](#) &str)
- **StringContains** ([Contains](#) cntn)
- bool **check** (const [String](#) &str)
- **operator const String &** () const
- const char \* **c\_str** () const

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 >:



### Additional Inherited Members

#### Static Public Member Functions inherited from

[doctest::detail::StringMakerBase< \[detail::has\\\_insertion\\\_operator< T >::value||detail::types::is\\\_pointer\]\(#\)](#)

- 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

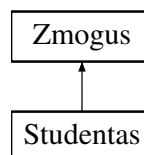
- `template<typename T>`  
static [String convert](#) (const DOCTEST\_REF\_WRAP(T) in)
- static [String convert](#) (const DOCTEST\_REF\_WRAP(T))

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 metodash)
- 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

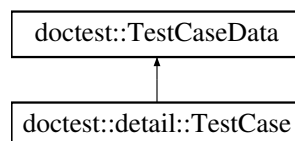
- [String](#) **m\_name**
- const char \* **m\_file**
- int **m\_line**

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](#)**

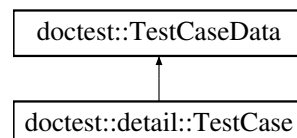
- [String](#) **m\_file**
- unsigned **m\_line**
- const char \* **m\_name**
- const char \* **m\_test\_suite**
- const char \* **m\_description**
- bool **m\_skip**
- bool **m\_no\_breaks**
- bool **m\_no\_output**
- bool **m\_may\_fail**
- bool **m\_should\_fail**
- int **m\_expected\_failures**
- 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:

**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

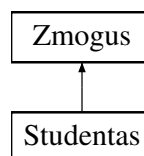
- `const_reverse_iterator` **crend** () const noexcept
- `bool` **empty** () const noexcept
- `size_type` **size** () const noexcept
- `size_type` **capacity** () const noexcept
- `size_type` **getReallocations** () const
- `size_type` **max\_size** () const noexcept
- `void` **reserve** (size\_type 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

```
00001 #ifndef FUNKCIJOS_H
00002 #define FUNKCIJOS_H
00003
00004 #include "Studentas.h"
00005 #include <string>
00006 #include "vector.h"
00007 #include <iostream>
00008 #include <fstream>
00009 #include <algorithm>
00010 #include <iomanip>
00011 #include <cstdlib>
00012 #include <ctime>
00013 #include <cctype>
00014 #include <sstream>
00015 #include <limits>
00016 #include <stdexcept>
00017 #include <chrono>
00018 #include <vector>
00019
00020 // Funkcijos, kurios dirba su studentų sąrašu
00021
00022 bool arTinkamasVardas(const std::string& tekstas);
00023 bool arTinkamasPazymys(int& pazymys);
00024
00025 double skaiciuotiVidurki(const Vector<int>& pazymiai);
00026 double skaiciuotiMediana(Vector<int> pazymiai);
00027
00028 // Dirba su vienu Studentas objektu
00029 void skaiciuotiGalutiniBala(Studentas& studentas, char metodas);
00030 void generuotiPazymius(Studentas& studentas, int kiek);
00031
00032 // Dirba su studentų vektoriumi
00033 void generuotiStudentus(Vector<Studentas>& studentai, int kiek, int ndSk);
00034 void spausdintiRezultatus(const Vector<Studentas>& studentai, std::ostream& out);
00035 void nuskaitytiIsFailo(Vector<Studentas>& studentai, const std::string& failoVardas);
00036 void rikiuotiStudentus(Vector<Studentas>& studentai, char kriterijus);
00037 void generuotiFaila(const std::string& failoPavadinimas, int studentuKiekis, int ndSk, char metodas);
00038 void padalintiStudentus(const Vector<Studentas>& studentai, Vector<Studentas>& vargsiai,
00039                          Vector<Studentas>& kietiakai);
00039 void spausdintiStudentusIFaila(const Vector<Studentas>& studentai, const std::string& failoVardas);
00040 void apdorotiFaila(const std::string& failoVardas, char metodas);
00041
00042 void vykdytiPrograma(); // Pagrindinė funkcija programos valdymui
00043
00044 #endif
```

### 6.2 studentas.h

```
00001 #ifndef STUDENTAS_H
00002 #define STUDENTAS_H
00003
00004 #include <ostream>
00005 #include <iostream>
```

```

00006 #include <string>
00007 #include "vector.h"
00008 #include <algorithm>
00009 #include <numeric>
00010 #include "zmogus.h"
00011
00012 class Studentas : public Zmogus {
00013 private:
00014     Vector<int> nd_;
00015     int egzaminas_;
00016     double galutinis_;
00017
00018 public:
00019     Studentas(); // tuščias konstruktorius
00020     Studentas(const std::string& vardas, const std::string& pavarde, const Vector<int>& nd, int
egzaminas);
00021     Studentas(const Studentas& other); // Copy constructor
00022     Studentas(Studentas&& other) noexcept; // Move constructor
00023     ~Studentas(); // Destruktorius
00024     Studentas& operator=(const Studentas& other); // Copy assignment
00025     Studentas& operator=(Studentas&& other) noexcept; // Move assignment
00026
00027     // Getteriai
00028     int egzaminas() const;
00029     double galutinis() const;
00030     const Vector<int>& nd() const;
00031
00032     // Setteriai
00033     void setEgzaminas(int egzaminas);
00034     void pridetiND(int pazymys);
00035
00036     // Metodai
00037     void skaiciuotiGalutini(char metodas);
00038     void generuotiPazymius(int kiek);
00039     std::istream& read(std::istream& is);
00040     std::ostream& spausdinti(std::ostream& os) const override;
00041
00042     // I/O operatoriai
00043     friend std::ostream& operator<<(std::ostream& os, const Studentas& s);
00044     friend std::istream& operator>>(std::istream& is, Studentas& s);
00045
00046     // Friend funkcijos palyginimui
00047     friend bool compareVardas(const Studentas& a, const Studentas& b);
00048     friend bool comparePavarde(const Studentas& a, const Studentas& b);
00049     friend bool compareGalutinis(const Studentas& a, const Studentas& b);
00050 };
00051
00052 // Deklaracijos palyginimui
00053 bool compareVardas(const Studentas& a, const Studentas& b);
00054 bool comparePavarde(const Studentas& a, const Studentas& b);
00055 bool compareGalutinis(const Studentas& a, const Studentas& b);
00056 #endif

```

## 6.3 doctest.h

```

00001 // ===== lgtm
00002 // == DO NOT MODIFY THIS FILE BY HAND - IT IS AUTO GENERATED BY CMAKE! ==
00003 // =====
00004 //
00005 // doctest.h - the lightest feature-rich C++ single-header testing framework for unit tests and TDD
00006 //
00007 // Copyright (c) 2016-2023 Viktor Kirilov
00008 //
00009 // Distributed under the MIT Software License
00010 // See accompanying file LICENSE.txt or copy at
00011 // https://opensource.org/licenses/MIT
00012 //
00013 // The documentation can be found at the library's page:
00014 // https://github.com/doctest/doctest/blob/master/doc/markdown/readme.md
00015 //
00016 // =====
00017 // =====
00018 // =====
00019 //
00020 // The library is heavily influenced by Catch - https://github.com/catchorg/Catch2
00021 // which uses the Boost Software License - Version 1.0
00022 // see here - https://github.com/catchorg/Catch2/blob/master/LICENSE.txt
00023 //
00024 // The concept of subcases (sections in Catch) and expression decomposition are from there.
00025 // Some parts of the code are taken directly:
00026 // - stringification - the detection of "ostream& operator<<(ostream&, const T&)" and StringMaker<>
00027 // - the Approx() helper class for floating point comparison

```

```

00028 // - colors in the console
00029 // - breaking into a debugger
00030 // - signal / SEH handling
00031 // - timer
00032 // - XmlWriter class - thanks to Phil Nash for allowing the direct reuse (AKA copy/paste)
00033 //
00034 // The expression decomposing templates are taken from lest - https://github.com/martinmoene/lest
00035 // which uses the Boost Software License - Version 1.0
00036 // see here - https://github.com/martinmoene/lest/blob/master/LICENSE.txt
00037 //
00038 // =====
00039 // =====
00040 // =====
00041
00042 #ifndef DOCTEST_LIBRARY_INCLUDED
00043 #define DOCTEST_LIBRARY_INCLUDED
00044
00045 // =====
00046 // == VERSION =====
00047 // =====
00048
00049 #define DOCTEST_VERSION_MAJOR 2
00050 #define DOCTEST_VERSION_MINOR 4
00051 #define DOCTEST_VERSION_PATCH 12
00052
00053 // util we need here
00054 #define DOCTEST_TOSTR_IMPL(x) #x
00055 #define DOCTEST_TOSTR(x) DOCTEST_TOSTR_IMPL(x)
00056
00057 #define DOCTEST_VERSION_STR                                     \
00058     DOCTEST_TOSTR(DOCTEST_VERSION_MAJOR) "."                \
00059     DOCTEST_TOSTR(DOCTEST_VERSION_MINOR) "."                 \
00060     DOCTEST_TOSTR(DOCTEST_VERSION_PATCH)                     \
00061
00062 #define DOCTEST_VERSION                                       \
00063     (DOCTEST_VERSION_MAJOR * 10000 + DOCTEST_VERSION_MINOR * 100 + DOCTEST_VERSION_PATCH)
00064
00065 // =====
00066 // == COMPILER VERSION =====
00067 // =====
00068
00069 // ideas for the version stuff are taken from here: https://github.com/cxxstuff/cxx_detect
00070
00071 #ifndef _MSC_VER
00072 #define DOCTEST_CPLUSPLUS _MSVC_LANG
00073 #else
00074 #define DOCTEST_CPLUSPLUS __cplusplus
00075 #endif
00076
00077 #define DOCTEST_COMPILER(MAJOR, MINOR, PATCH) ((MAJOR)*10000000 + (MINOR)*100000 + (PATCH))
00078
00079 // GCC/Clang and GCC/MSVC are mutually exclusive, but Clang/MSVC are not because of clang-cl...
00080 #if defined(_MSC_VER) && defined(_MSC_FULL_VER)
00081 #if _MSC_VER == _MSC_FULL_VER / 10000
00082 #define DOCTEST_MSVC DOCTEST_COMPILER(_MSC_VER / 100, _MSC_VER % 100, _MSC_FULL_VER % 10000)
00083 #else // MSVC
00084 #define DOCTEST_MSVC                                     \
00085     DOCTEST_COMPILER(_MSC_VER / 100, (_MSC_FULL_VER / 100000) % 100, _MSC_FULL_VER % 100000)
00086 #endif // MSVC
00087 #endif // MSVC
00088 #if defined(__clang__) && defined(__clang_minor__) && defined(__clang_patchlevel__)
00089 #define DOCTEST_CLANG DOCTEST_COMPILER(__clang_major__, __clang_minor__, __clang_patchlevel__)
00090 #elif defined(__GNUC__) && defined(__GNUC_MINOR__) && defined(__GNUC_PATCHLEVEL__) &&
00091     !defined(__INTEL_COMPILER)
00092 #define DOCTEST_GCC DOCTEST_COMPILER(__GNUC__, __GNUC_MINOR__, __GNUC_PATCHLEVEL__)
00093 #endif // GCC
00094 #if defined(__INTEL_COMPILER)
00095 #define DOCTEST_ICC DOCTEST_COMPILER(__INTEL_COMPILER / 100, __INTEL_COMPILER % 100, 0)
00096 #endif // ICC
00097
00098 #ifndef DOCTEST_MSVC
00099 #define DOCTEST_MSVC 0
00100 #endif // DOCTEST_MSVC
00101 #ifndef DOCTEST_CLANG
00102 #define DOCTEST_CLANG 0
00103 #endif // DOCTEST_CLANG
00104 #ifndef DOCTEST_GCC
00105 #define DOCTEST_GCC 0
00106 #endif // DOCTEST_GCC
00107 #ifndef DOCTEST_ICC
00108 #define DOCTEST_ICC 0
00109 #endif // DOCTEST_ICC
00110
00111 // =====
00112 // == COMPILER WARNINGS HELPERS =====
00113 // =====
00114

```

```

00115 #if DOCTEST_CLANG && !DOCTEST_ICC
00116 #define DOCTEST_PRAGMA_TO_STR(x) _Pragma(#x)
00117 #define DOCTEST_CLANG_SUPPRESS_WARNING_PUSH _Pragma("clang diagnostic push")
00118 #define DOCTEST_CLANG_SUPPRESS_WARNING(w) DOCTEST_PRAGMA_TO_STR(clang diagnostic ignored w)
00119 #define DOCTEST_CLANG_SUPPRESS_WARNING_POP _Pragma("clang diagnostic pop")
00120 #define DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH(w) \
00121     DOCTEST_CLANG_SUPPRESS_WARNING_PUSH DOCTEST_CLANG_SUPPRESS_WARNING(w)
00122 #else // DOCTEST_CLANG
00123 #define DOCTEST_CLANG_SUPPRESS_WARNING_PUSH
00124 #define DOCTEST_CLANG_SUPPRESS_WARNING(w)
00125 #define DOCTEST_CLANG_SUPPRESS_WARNING_POP
00126 #define DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH(w)
00127 #endif // DOCTEST_CLANG
00128
00129 #if DOCTEST_GCC
00130 #define DOCTEST_PRAGMA_TO_STR(x) _Pragma(#x)
00131 #define DOCTEST_GCC_SUPPRESS_WARNING_PUSH _Pragma("GCC diagnostic push")
00132 #define DOCTEST_GCC_SUPPRESS_WARNING(w) DOCTEST_PRAGMA_TO_STR(GCC diagnostic ignored w)
00133 #define DOCTEST_GCC_SUPPRESS_WARNING_POP _Pragma("GCC diagnostic pop")
00134 #define DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH(w) \
00135     DOCTEST_GCC_SUPPRESS_WARNING_PUSH DOCTEST_GCC_SUPPRESS_WARNING(w)
00136 #else // DOCTEST_GCC
00137 #define DOCTEST_GCC_SUPPRESS_WARNING_PUSH
00138 #define DOCTEST_GCC_SUPPRESS_WARNING(w)
00139 #define DOCTEST_GCC_SUPPRESS_WARNING_POP
00140 #define DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH(w)
00141 #endif // DOCTEST_GCC
00142
00143 #if DOCTEST_MSVC
00144 #define DOCTEST_MSVC_SUPPRESS_WARNING_PUSH __pragma(warning(push))
00145 #define DOCTEST_MSVC_SUPPRESS_WARNING(w) __pragma(warning(disable : w))
00146 #define DOCTEST_MSVC_SUPPRESS_WARNING_POP __pragma(warning(pop))
00147 #define DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(w) \
00148     DOCTEST_MSVC_SUPPRESS_WARNING_PUSH DOCTEST_MSVC_SUPPRESS_WARNING(w)
00149 #else // DOCTEST_MSVC
00150 #define DOCTEST_MSVC_SUPPRESS_WARNING_PUSH
00151 #define DOCTEST_MSVC_SUPPRESS_WARNING(w)
00152 #define DOCTEST_MSVC_SUPPRESS_WARNING_POP
00153 #define DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(w)
00154 #endif // DOCTEST_MSVC
00155
00156 // =====
00157 // == COMPILER WARNINGS ==
00158 // =====
00159
00160 // both the header and the implementation suppress all of these,
00161 // so it only makes sense to aggregate them like so
00162 #define DOCTEST_SUPPRESS_COMMON_WARNINGS_PUSH \
00163     DOCTEST_CLANG_SUPPRESS_WARNING_PUSH \
00164     DOCTEST_CLANG_SUPPRESS_WARNING("-Wunknown-pragmas") \
00165     DOCTEST_CLANG_SUPPRESS_WARNING("-Wweak-vmtables") \
00166     DOCTEST_CLANG_SUPPRESS_WARNING("-Wpadded") \
00167     DOCTEST_CLANG_SUPPRESS_WARNING("-Wmissing-prototypes") \
00168     DOCTEST_CLANG_SUPPRESS_WARNING("-Wc++98-compat") \
00169     DOCTEST_CLANG_SUPPRESS_WARNING("-Wc++98-compat-pedantic") \
00170 \
00171     DOCTEST_GCC_SUPPRESS_WARNING_PUSH \
00172     DOCTEST_GCC_SUPPRESS_WARNING("-Wunknown-pragmas") \
00173     DOCTEST_GCC_SUPPRESS_WARNING("-Wpragmas") \
00174     DOCTEST_GCC_SUPPRESS_WARNING("-Weffc++") \
00175     DOCTEST_GCC_SUPPRESS_WARNING("-Wstrict-overflow") \
00176     DOCTEST_GCC_SUPPRESS_WARNING("-Wstrict-aliasing") \
00177     DOCTEST_GCC_SUPPRESS_WARNING("-Wmissing-declarations") \
00178     DOCTEST_GCC_SUPPRESS_WARNING("-Wuseless-cast") \
00179     DOCTEST_GCC_SUPPRESS_WARNING("-Wnoexcept") \
00180 \
00181     DOCTEST_MSVC_SUPPRESS_WARNING_PUSH \
00182     /* these 4 also disabled globally via cmake: */ \
00183     DOCTEST_MSVC_SUPPRESS_WARNING(4514) /* unreferenced inline function has been removed */ \
00184     DOCTEST_MSVC_SUPPRESS_WARNING(4571) /* SEH related */ \
00185     DOCTEST_MSVC_SUPPRESS_WARNING(4710) /* function not inlined */ \
00186     DOCTEST_MSVC_SUPPRESS_WARNING(4711) /* function selected for inline expansion */ \
00187     /* common ones */ \
00188     DOCTEST_MSVC_SUPPRESS_WARNING(4616) /* invalid compiler warning */ \
00189     DOCTEST_MSVC_SUPPRESS_WARNING(4619) /* invalid compiler warning */ \
00190     DOCTEST_MSVC_SUPPRESS_WARNING(4996) /* The compiler encountered a deprecated declaration */ \
00191     DOCTEST_MSVC_SUPPRESS_WARNING(4706) /* assignment within conditional expression */ \
00192     DOCTEST_MSVC_SUPPRESS_WARNING(4512) /* 'class' : assignment operator could not be generated */ \
00193     DOCTEST_MSVC_SUPPRESS_WARNING(4127) /* conditional expression is constant */ \
00194     DOCTEST_MSVC_SUPPRESS_WARNING(4820) /* padding */ \
00195     DOCTEST_MSVC_SUPPRESS_WARNING(4625) /* copy constructor was implicitly deleted */ \
00196     DOCTEST_MSVC_SUPPRESS_WARNING(4626) /* assignment operator was implicitly deleted */ \
00197     DOCTEST_MSVC_SUPPRESS_WARNING(5027) /* move assignment operator implicitly deleted */ \
00198     DOCTEST_MSVC_SUPPRESS_WARNING(5026) /* move constructor was implicitly deleted */ \
00199     DOCTEST_MSVC_SUPPRESS_WARNING(4640) /* construction of local static object not thread-safe */ \
00200     DOCTEST_MSVC_SUPPRESS_WARNING(5045) /* Spectre mitigation for memory load */ \
00201     DOCTEST_MSVC_SUPPRESS_WARNING(5264) /* 'variable-name': 'const' variable is not used */

```

```

00202      /* static analysis */
00203      DOCTEST_MSVC_SUPPRESS_WARNING(26439) /* Function may not throw. Declare it 'noexcept' */
00204      DOCTEST_MSVC_SUPPRESS_WARNING(26495) /* Always initialize a member variable */
00205      DOCTEST_MSVC_SUPPRESS_WARNING(26451) /* Arithmetic overflow ... */
00206      DOCTEST_MSVC_SUPPRESS_WARNING(26444) /* Avoid unnamed objects with custom ctor and dtor... */
00207      DOCTEST_MSVC_SUPPRESS_WARNING(26812) /* Prefer 'enum class' over 'enum' */
00208
00209      #define DOCTEST_SUPPRESS_COMMON_WARNINGS_POP
00210      DOCTEST_CLANG_SUPPRESS_WARNING_POP
00211      DOCTEST_GCC_SUPPRESS_WARNING_POP
00212      DOCTEST_MSVC_SUPPRESS_WARNING_POP
00213
00214      DOCTEST_SUPPRESS_COMMON_WARNINGS_PUSH
00215
00216      DOCTEST_CLANG_SUPPRESS_WARNING_PUSH
00217      DOCTEST_CLANG_SUPPRESS_WARNING("-Wnon-virtual-dtor")
00218      DOCTEST_CLANG_SUPPRESS_WARNING("-Wdeprecated")
00219
00220      DOCTEST_GCC_SUPPRESS_WARNING_PUSH
00221      DOCTEST_GCC_SUPPRESS_WARNING("-Wctor-dtor-privacy")
00222      DOCTEST_GCC_SUPPRESS_WARNING("-Wnon-virtual-dtor")
00223      DOCTEST_GCC_SUPPRESS_WARNING("-Wsign-promo")
00224
00225      DOCTEST_MSVC_SUPPRESS_WARNING_PUSH
00226      DOCTEST_MSVC_SUPPRESS_WARNING(4623) // default constructor was implicitly defined as deleted
00227
00228      #define DOCTEST_MAKE_STD_HEADERS_CLEAN_FROM_WARNINGS_ON_WALL_BEGIN
00229      DOCTEST_MSVC_SUPPRESS_WARNING_PUSH
00230      DOCTEST_MSVC_SUPPRESS_WARNING(4548) /* before comma no effect; expected side - effect */
00231      DOCTEST_MSVC_SUPPRESS_WARNING(4265) /* virtual functions, but destructor is not virtual */
00232      DOCTEST_MSVC_SUPPRESS_WARNING(4986) /* exception specification does not match previous */
00233      DOCTEST_MSVC_SUPPRESS_WARNING(4350) /* 'member1' called instead of 'member2' */
00234      DOCTEST_MSVC_SUPPRESS_WARNING(4668) /* not defined as a preprocessor macro */
00235      DOCTEST_MSVC_SUPPRESS_WARNING(4365) /* signed/unsigned mismatch */
00236      DOCTEST_MSVC_SUPPRESS_WARNING(4774) /* format string not a string literal */
00237      DOCTEST_MSVC_SUPPRESS_WARNING(4820) /* padding */
00238      DOCTEST_MSVC_SUPPRESS_WARNING(4625) /* copy constructor was implicitly deleted */
00239      DOCTEST_MSVC_SUPPRESS_WARNING(4626) /* assignment operator was implicitly deleted */
00240      DOCTEST_MSVC_SUPPRESS_WARNING(5027) /* move assignment operator implicitly deleted */
00241      DOCTEST_MSVC_SUPPRESS_WARNING(5026) /* move constructor was implicitly deleted */
00242      DOCTEST_MSVC_SUPPRESS_WARNING(4623) /* default constructor was implicitly deleted */
00243      DOCTEST_MSVC_SUPPRESS_WARNING(5039) /* pointer to pot. throwing function passed to extern C */
00244      DOCTEST_MSVC_SUPPRESS_WARNING(5045) /* Spectre mitigation for memory load */
00245      DOCTEST_MSVC_SUPPRESS_WARNING(5105) /* macro producing 'defined' has undefined behavior */
00246      DOCTEST_MSVC_SUPPRESS_WARNING(4738) /* storing float result in memory, loss of performance */
00247      DOCTEST_MSVC_SUPPRESS_WARNING(5262) /* implicit fall-through */
00248
00249      #define DOCTEST_MAKE_STD_HEADERS_CLEAN_FROM_WARNINGS_ON_WALL_END DOCTEST_MSVC_SUPPRESS_WARNING_POP
00250
00251      // =====
00252      // == FEATURE DETECTION =====
00253      // =====
00254
00255      // general compiler feature support table: https://en.cppreference.com/w/cpp/compiler_support
00256      // MSVC C++11 feature support table: https://msdn.microsoft.com/en-us/library/hh567368.aspx
00257      // GCC C++11 feature support table: https://gcc.gnu.org/projects/cxx-status.html
00258      // MSVC version table:
00259      // https://en.wikipedia.org/wiki/Microsoft_Visual_C%2B%2B#Internal_version_numbering
00260      // MSVC++ 14.3 (17) _MSC_VER == 1930 (Visual Studio 2022)
00261      // MSVC++ 14.2 (16) _MSC_VER == 1920 (Visual Studio 2019)
00262      // MSVC++ 14.1 (15) _MSC_VER == 1910 (Visual Studio 2017)
00263      // MSVC++ 14.0 _MSC_VER == 1900 (Visual Studio 2015)
00264      // MSVC++ 12.0 _MSC_VER == 1800 (Visual Studio 2013)
00265      // MSVC++ 11.0 _MSC_VER == 1700 (Visual Studio 2012)
00266      // MSVC++ 10.0 _MSC_VER == 1600 (Visual Studio 2010)
00267      // MSVC++ 9.0 _MSC_VER == 1500 (Visual Studio 2008)
00268      // MSVC++ 8.0 _MSC_VER == 1400 (Visual Studio 2005)
00269
00270      // Universal Windows Platform support
00271      #if defined(WINAPI_FAMILY) && (WINAPI_FAMILY == WINAPI_FAMILY_APP)
00272      #define DOCTEST_CONFIG_NO_WINDOWS_SEH
00273      #endif // WINAPI_FAMILY
00274      #if DOCTEST_MSVC && !defined(DOCTEST_CONFIG_WINDOWS_SEH)
00275      #define DOCTEST_CONFIG_WINDOWS_SEH
00276      #endif // MSVC
00277      #if defined(DOCTEST_CONFIG_NO_WINDOWS_SEH) && defined(DOCTEST_CONFIG_WINDOWS_SEH)
00278      #undef DOCTEST_CONFIG_WINDOWS_SEH
00279      #endif // DOCTEST_CONFIG_NO_WINDOWS_SEH
00280
00281      #if !defined(_WIN32) && !defined(__QNX__) && !defined(DOCTEST_CONFIG_POSIX_SIGNALS) &&
00282          !defined(__EMSCRIPTEN__) && !defined(__wasi__)
00283      #define DOCTEST_CONFIG_POSIX_SIGNALS
00284      #endif // _WIN32
00285      #if defined(DOCTEST_CONFIG_NO_POSIX_SIGNALS) && defined(DOCTEST_CONFIG_POSIX_SIGNALS)
00286      #undef DOCTEST_CONFIG_POSIX_SIGNALS
00287      #endif // DOCTEST_CONFIG_NO_POSIX_SIGNALS
00288

```

```

00289 #ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
00290 #if !defined(__cpp_exceptions) && !defined(__EXCEPTIONS) && !defined(_CPPUNWIND) \
00291     || defined(__wasi__)
00292 #define DOCTEST_CONFIG_NO_EXCEPTIONS
00293 #endif // no exceptions
00294 #endif // DOCTEST_CONFIG_NO_EXCEPTIONS
00295
00296 #ifdef DOCTEST_CONFIG_NO_EXCEPTIONS_BUT_WITH_ALL_ASSERTS
00297 #ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
00298 #define DOCTEST_CONFIG_NO_EXCEPTIONS
00299 #endif // DOCTEST_CONFIG_NO_EXCEPTIONS
00300 #endif // DOCTEST_CONFIG_NO_EXCEPTIONS_BUT_WITH_ALL_ASSERTS
00301
00302 #if defined(DOCTEST_CONFIG_NO_EXCEPTIONS) && !defined(DOCTEST_CONFIG_NO_TRY_CATCH_IN_ASSERTS)
00303 #define DOCTEST_CONFIG_NO_TRY_CATCH_IN_ASSERTS
00304 #endif // DOCTEST_CONFIG_NO_EXCEPTIONS && !DOCTEST_CONFIG_NO_TRY_CATCH_IN_ASSERTS
00305
00306 #ifdef __wasi__
00307 #define DOCTEST_CONFIG_NO_MULTITHREADING
00308 #endif
00309
00310 #if defined(DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN) && !defined(DOCTEST_CONFIG_IMPLEMENT)
00311 #define DOCTEST_CONFIG_IMPLEMENT
00312 #endif // DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
00313
00314 #if defined(_WIN32) || defined(__CYGWIN__)
00315 #if DOCTEST_MSVC
00316 #define DOCTEST_SYMBOL_EXPORT __declspec(dllexport)
00317 #define DOCTEST_SYMBOL_IMPORT __declspec(dllimport)
00318 #else // MSVC
00319 #define DOCTEST_SYMBOL_EXPORT __attribute__((dllexport))
00320 #define DOCTEST_SYMBOL_IMPORT __attribute__((dllimport))
00321 #endif // MSVC
00322 #else // _WIN32
00323 #define DOCTEST_SYMBOL_EXPORT __attribute__((visibility("default")))
00324 #define DOCTEST_SYMBOL_IMPORT
00325 #endif // _WIN32
00326
00327 #ifdef DOCTEST_CONFIG_IMPLEMENTATION_IN_DLL
00328 #ifdef DOCTEST_CONFIG_IMPLEMENT
00329 #define DOCTEST_INTERFACE DOCTEST_SYMBOL_EXPORT
00330 #else // DOCTEST_CONFIG_IMPLEMENT
00331 #define DOCTEST_INTERFACE DOCTEST_SYMBOL_IMPORT
00332 #endif // DOCTEST_CONFIG_IMPLEMENT
00333 #else // DOCTEST_CONFIG_IMPLEMENTATION_IN_DLL
00334 #define DOCTEST_INTERFACE
00335 #endif // DOCTEST_CONFIG_IMPLEMENTATION_IN_DLL
00336
00337 // needed for extern template instantiations
00338 // see https://github.com/fmtlib/fmt/issues/2228
00339 #if DOCTEST_MSVC
00340 #define DOCTEST_INTERFACE_DECL
00341 #define DOCTEST_INTERFACE_DEF DOCTEST_INTERFACE
00342 #else // DOCTEST_MSVC
00343 #define DOCTEST_INTERFACE_DECL DOCTEST_INTERFACE
00344 #define DOCTEST_INTERFACE_DEF
00345 #endif // DOCTEST_MSVC
00346
00347 #define DOCTEST_EMPTY
00348
00349 #if DOCTEST_MSVC
00350 #define DOCTEST_NOINLINE __declspec(noinline)
00351 #define DOCTEST_UNUSED
00352 #define DOCTEST_ALIGNMENT(x)
00353 #elif DOCTEST_CLANG && DOCTEST_CLANG < DOCTEST_COMPILER(3, 5, 0)
00354 #define DOCTEST_NOINLINE
00355 #define DOCTEST_UNUSED
00356 #define DOCTEST_ALIGNMENT(x)
00357 #else
00358 #define DOCTEST_NOINLINE __attribute__((noinline))
00359 #define DOCTEST_UNUSED __attribute__((unused))
00360 #define DOCTEST_ALIGNMENT(x) __attribute__((aligned(x)))
00361 #endif
00362
00363 #ifdef DOCTEST_CONFIG_NO_CONTRADICTING_INLINE
00364 #define DOCTEST_INLINE_NOINLINE inline
00365 #else
00366 #define DOCTEST_INLINE_NOINLINE inline DOCTEST_NOINLINE
00367 #endif
00368
00369 #ifndef DOCTEST_NORETURN
00370 #if DOCTEST_MSVC && (DOCTEST_MSVC < DOCTEST_COMPILER(19, 0, 0))
00371 #define DOCTEST_NORETURN
00372 #else // DOCTEST_MSVC
00373 #define DOCTEST_NORETURN [[noreturn]]
00374 #endif // DOCTEST_MSVC
00375 #endif // DOCTEST_NORETURN

```

```

00376
00377 #ifndef DOCTEST_NOEXCEPT
00378 #if DOCTEST_MSVC && (DOCTEST_MSVC < DOCTEST_COMPILER(19, 0, 0))
00379 #define DOCTEST_NOEXCEPT
00380 #else // DOCTEST_MSVC
00381 #define DOCTEST_NOEXCEPT noexcept
00382 #endif // DOCTEST_MSVC
00383 #endif // DOCTEST_NOEXCEPT
00384
00385 #ifndef DOCTEST_CONSTEXPR
00386 #if DOCTEST_MSVC && (DOCTEST_MSVC < DOCTEST_COMPILER(19, 0, 0))
00387 #define DOCTEST_CONSTEXPR const
00388 #define DOCTEST_CONSTEXPR_FUNC inline
00389 #else // DOCTEST_MSVC
00390 #define DOCTEST_CONSTEXPR constexpr
00391 #define DOCTEST_CONSTEXPR_FUNC constexpr
00392 #endif // DOCTEST_MSVC
00393 #endif // DOCTEST_CONSTEXPR
00394
00395 #ifndef DOCTEST_NO_SANITIZE_INTEGER
00396 #if DOCTEST_CLANG >= DOCTEST_COMPILER(3, 7, 0)
00397 #define DOCTEST_NO_SANITIZE_INTEGER __attribute__((no_sanitize("integer")))
00398 #else
00399 #define DOCTEST_NO_SANITIZE_INTEGER
00400 #endif
00401 #endif // DOCTEST_NO_SANITIZE_INTEGER
00402
00403 // =====
00404 // == FEATURE DETECTION END ==
00405 // =====
00406
00407 #define DOCTEST_DECLARE_INTERFACE(name)
00408     virtual ~name();
00409     name() = default;
00410     name(const name&) = delete;
00411     name(name&&) = delete;
00412     name& operator=(const name&) = delete;
00413     name& operator=(name&&) = delete;
00414
00415 #define DOCTEST_DEFINE_INTERFACE(name)
00416     name::~name() = default;
00417
00418 // internal macros for string concatenation and anonymous variable name generation
00419 #define DOCTEST_CAT_IMPL(s1, s2) s1##s2
00420 #define DOCTEST_CAT(s1, s2) DOCTEST_CAT_IMPL(s1, s2)
00421 #ifndef __COUNTER__ // not standard and may be missing for some compilers
00422 #define DOCTEST_ANONYMOUS(x) DOCTEST_CAT(x, __COUNTER__)
00423 #else // __COUNTER__
00424 #define DOCTEST_ANONYMOUS(x) DOCTEST_CAT(x, __LINE__)
00425 #endif // __COUNTER__
00426
00427 #ifndef DOCTEST_CONFIG_ASSERTION_PARAMETERS_BY_VALUE
00428 #define DOCTEST_REF_WRAP(x) x&
00429 #else // DOCTEST_CONFIG_ASSERTION_PARAMETERS_BY_VALUE
00430 #define DOCTEST_REF_WRAP(x) x
00431 #endif // DOCTEST_CONFIG_ASSERTION_PARAMETERS_BY_VALUE
00432
00433 // not using __APPLE__ because... this is how Catch does it
00434 #ifndef __MAC_OS_X_VERSION_MIN_REQUIRED
00435 #define DOCTEST_PLATFORM_MAC
00436 #elif defined(__IPHONE_OS_VERSION_MIN_REQUIRED)
00437 #define DOCTEST_PLATFORM_IPHONE
00438 #elif defined(__WIN32)
00439 #define DOCTEST_PLATFORM_WINDOWS
00440 #elif defined(__wasi__)
00441 #define DOCTEST_PLATFORM_WASI
00442 #else // DOCTEST_PLATFORM
00443 #define DOCTEST_PLATFORM_LINUX
00444 #endif // DOCTEST_PLATFORM
00445
00446 namespace doctest { namespace detail {
00447     static DOCTEST_CONSTEXPR int consume(const int*, int) noexcept { return 0; }
00448 }}
00449
00450 #define DOCTEST_GLOBAL_NO_WARNINGS(var, ...)
00451     DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wglobal-constructors")
00452     static const int var = doctest::detail::consume(&var, __VA_ARGS__);
00453     DOCTEST_CLANG_SUPPRESS_WARNING_POP
00454
00455 #ifndef DOCTEST_BREAK_INTO_DEBUGGER
00456 // should probably take a look at https://github.com/scottt/debugbreak
00457 #ifndef DOCTEST_PLATFORM_LINUX
00458 #if defined(__GNUC__) && (defined(__i386) || defined(__x86_64))
00459 // Break at the location of the failing check if possible
00460 #define DOCTEST_BREAK_INTO_DEBUGGER() __asm__("int $3\n" : :) // NOLINT(hicpp-no-assembler)
00461 #else
00462 #include <signal.h>

```



```

00463 #define DOCTEST_BREAK_INTO_DEBUGGER() raise(SIGTRAP)
00464 #endif
00465 #elif defined(DOCTEST_PLATFORM_MAC)
00466 #if defined(__x86_64__) || defined(__x86_64__) || defined(__amd64__) || defined(__i386)
00467 #define DOCTEST_BREAK_INTO_DEBUGGER() __asm__("int $3\n" : :) // NOLINT(hicpp-no-assembler)
00468 #elif defined(__ppc__) || defined(__ppc64__)
00469 // https://www.cocoawithlove.com/2008/03/break-into-debugger.html
00470 #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)
00471 #else
00472 #define DOCTEST_BREAK_INTO_DEBUGGER() __asm__("brk #0"); // NOLINT(hicpp-no-assembler)
00473 #endif
00474 #elif DOCTEST_MSVC
00475 #define DOCTEST_BREAK_INTO_DEBUGGER() __debugbreak()
00476 #elif defined(__MINGW32__)
00477 DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH("-Wredundant-decls")
00478 extern "C" __declspec(dllimport) void __stdcall DebugBreak();
00479 DOCTEST_GCC_SUPPRESS_WARNING_POP
00480 #define DOCTEST_BREAK_INTO_DEBUGGER() ::DebugBreak()
00481 #else // linux
00482 #define DOCTEST_BREAK_INTO_DEBUGGER() (static_cast<void>(0))
00483 #endif // linux
00484 #endif // DOCTEST_BREAK_INTO_DEBUGGER
00485
00486 // this is kept here for backwards compatibility since the config option was changed
00487 #ifdef DOCTEST_CONFIG_USE_IOFWD
00488 #ifndef DOCTEST_CONFIG_USE_STD_HEADERS
00489 #define DOCTEST_CONFIG_USE_STD_HEADERS
00490 #endif
00491 #endif // DOCTEST_CONFIG_USE_IOFWD
00492
00493 // for clang - always include ciso646 (which drags some std stuff) because
00494 // we want to check if we are using libc++ with the _LIBCPP_VERSION macro in
00495 // which case we don't want to forward declare stuff from std - for reference:
00496 // https://github.com/doctest/doctest/issues/126
00497 // https://github.com/doctest/doctest/issues/356
00498 #if DOCTEST_CLANG
00499 #include <ciso646>
00500 #endif // clang
00501
00502 #ifdef _LIBCPP_VERSION
00503 #ifndef DOCTEST_CONFIG_USE_STD_HEADERS
00504 #define DOCTEST_CONFIG_USE_STD_HEADERS
00505 #endif
00506 #endif // _LIBCPP_VERSION
00507
00508 #ifdef DOCTEST_CONFIG_USE_STD_HEADERS
00509 #ifndef DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
00510 #define DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
00511 #endif // DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
00512 DOCTEST_MAKE_STD_HEADERS_CLEAN_FROM_WARNINGS_ON_WALL_BEGIN
00513 #include <cstdint>
00514 #include <ostream>
00515 #include <istream>
00516 DOCTEST_MAKE_STD_HEADERS_CLEAN_FROM_WARNINGS_ON_WALL_END
00517 #else // DOCTEST_CONFIG_USE_STD_HEADERS
00518
00519 // Forward declaring 'X' in namespace std is not permitted by the C++ Standard.
00520 DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4643)
00521
00522 namespace std { // NOLINT(cert-dcl58-cpp)
00523 typedef decltype(nullptr) nullptr_t; // NOLINT(modernize-use-using)
00524 typedef decltype(sizeof(void*)) size_t; // NOLINT(modernize-use-using)
00525 template <class charT>
00526 struct char_traits;
00527 template <>
00528 struct char_traits<char>;
00529 template <class charT, class traits>
00530 class basic_ostream; // NOLINT(fuchsia-virtual-inheritance)
00531 typedef basic_ostream<char, char_traits<char>> ostream; // NOLINT(modernize-use-using)
00532 template<class traits>
00533 // NOLINTNEXTLINE
00534 basic_ostream<char, traits>& operator<<(basic_ostream<char, traits>&, const char*);
00535 template <class charT, class traits>
00536 class basic_istream;
00537 typedef basic_istream<char, char_traits<char>> istream; // NOLINT(modernize-use-using)
00538 template <class... Types>
00539 class tuple;
00540 #if DOCTEST_MSVC >= DOCTEST_COMPILER(19, 20, 0)
00541 // see this issue on why this is needed: https://github.com/doctest/doctest/issues/183
00542 template <class Ty>
00543 class allocator;
00544 template <class Elem, class Traits, class Alloc>
00545 class basic_string;
00546 using string = basic_string<char, char_traits<char>, allocator<char>;
00547 #endif // VS 2019
00548 } // namespace std

```



```

00549
00550 DOCTEST_MSVC_SUPPRESS_WARNING_POP
00551
00552 #endif // DOCTEST_CONFIG_USE_STD_HEADERS
00553
00554 #ifndef DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
00555 #include <type_traits>
00556 #endif // DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
00557
00558 namespace doctest {
00559
00560 using std::size_t;
00561
00562 DOCTEST_INTERFACE extern bool is_running_in_test;
00563
00564 #ifndef DOCTEST_CONFIG_STRING_SIZE_TYPE
00565 #define DOCTEST_CONFIG_STRING_SIZE_TYPE unsigned
00566 #endif
00567
00568 // A 24 byte string class (can be as small as 17 for x64 and 13 for x86) that can hold strings with
// length
00569 // of up to 23 chars on the stack before going on the heap - the last byte of the buffer is used for:
00570 // - "is small" bit - the highest bit - if "0" then it is small - otherwise its "1" (128)
00571 // - if small - capacity left before going on the heap - using the lowest 5 bits
00572 // - if small - 2 bits are left unused - the second and third highest ones
00573 // - if small - acts as a null terminator if strlen() is 23 (24 including the null terminator)
00574 // and the "is small" bit remains "0" ("as well as the capacity left") so its OK
00575 // Idea taken from this lecture about the string implementation of facebook/folly - fbstring
00576 // https://www.youtube.com/watch?v=kPR8h4-qZdk
00577 // TODO:
00578 // - optimizations - like not deleting memory unnecessarily in operator= and etc.
00579 // - resize/reserve/clear
00580 // - replace
00581 // - back/front
00582 // - iterator stuff
00583 // - find & friends
00584 // - push_back/pop_back
00585 // - assign/insert/erase
00586 // - relational operators as free functions - taking const char* as one of the params
00587 class DOCTEST_INTERFACE String
00588 {
00589 public:
00590     using size_type = DOCTEST_CONFIG_STRING_SIZE_TYPE;
00591
00592 private:
00593     static DOCTEST_CONSTEXPR size_type len = 24;
00594     static DOCTEST_CONSTEXPR size_type last = len - 1;
00595
00596     struct view // len should be more than sizeof(view) - because of the final byte for flags
00597     {
00598         char* ptr;
00599         size_type size;
00600         size_type capacity;
00601     };
00602
00603     union
00604     {
00605         char buf[len]; // NOLINT(*-avoid-c-arrays)
00606         view data;
00607     };
00608
00609     char* allocate(size_type sz);
00610
00611     bool isOnStack() const noexcept { return (buf[last] & 128) == 0; }
00612     void setOnHeap() noexcept;
00613     void setLast(size_type in = last) noexcept;
00614     void setSize(size_type sz) noexcept;
00615
00616     void copy(const String& other);
00617
00618 public:
00619     static DOCTEST_CONSTEXPR size_type npos = static_cast<size_type>(-1);
00620
00621     String() noexcept;
00622     ~String();
00623
00624     // cppcheck-suppress noExplicitConstructor
00625     String(const char* in);
00626     String(const char* in, size_type in_size);
00627
00628     String(std::istream& in, size_type in_size);
00629
00630     String(const String& other);
00631     String& operator=(const String& other);
00632
00633     String& operator+=(const String& other);
00634

```

```

00635     String(String&& other) noexcept;
00636     String& operator=(String&& other) noexcept;
00637
00638     char operator[](size_type i) const;
00639     char& operator[](size_type i);
00640
00641     // the only functions I'm willing to leave in the interface - available for inlining
00642     const char* c_str() const { return const_cast<String*>(this)->c_str(); } // NOLINT
00643     char* c_str() {
00644         if (isOnStack()) {
00645             return reinterpret_cast<char*>(buf);
00646         }
00647         return data.ptr;
00648     }
00649
00650     size_type size() const;
00651     size_type capacity() const;
00652
00653     String substr(size_type pos, size_type cnt = npos) &&;
00654     String substr(size_type pos, size_type cnt = npos) const &;
00655
00656     size_type find(char ch, size_type pos = 0) const;
00657     size_type rfind(char ch, size_type pos = npos) const;
00658
00659     int compare(const char* other, bool no_case = false) const;
00660     int compare(const String& other, bool no_case = false) const;
00661
00662     friend DOCTEST_INTERFACE std::ostream& operator<<(std::ostream& s, const String& in);
00663 };
00664
00665 DOCTEST_INTERFACE String operator+(const String& lhs, const String& rhs);
00666
00667 DOCTEST_INTERFACE bool operator==(const String& lhs, const String& rhs);
00668 DOCTEST_INTERFACE bool operator!=(const String& lhs, const String& rhs);
00669 DOCTEST_INTERFACE bool operator<(const String& lhs, const String& rhs);
00670 DOCTEST_INTERFACE bool operator>(const String& lhs, const String& rhs);
00671 DOCTEST_INTERFACE bool operator<=(const String& lhs, const String& rhs);
00672 DOCTEST_INTERFACE bool operator>=(const String& lhs, const String& rhs);
00673
00674 class DOCTEST_INTERFACE Contains {
00675 public:
00676     explicit Contains(const String& string);
00677
00678     bool checkWith(const String& other) const;
00679
00680     String string;
00681 };
00682
00683 DOCTEST_INTERFACE String toString(const Contains& in);
00684
00685 DOCTEST_INTERFACE bool operator==(const String& lhs, const Contains& rhs);
00686 DOCTEST_INTERFACE bool operator==(const Contains& lhs, const String& rhs);
00687 DOCTEST_INTERFACE bool operator!=(const String& lhs, const Contains& rhs);
00688 DOCTEST_INTERFACE bool operator!=(const Contains& lhs, const String& rhs);
00689
00690 namespace Color {
00691     enum Enum
00692     {
00693         None = 0,
00694         White,
00695         Red,
00696         Green,
00697         Blue,
00698         Cyan,
00699         Yellow,
00700         Grey,
00701
00702         Bright = 0x10,
00703
00704         BrightRed   = Bright | Red,
00705         BrightGreen = Bright | Green,
00706         LightGrey   = Bright | Grey,
00707         BrightWhite = Bright | White
00708     };
00709
00710     DOCTEST_INTERFACE std::ostream& operator<<(std::ostream& s, Color::Enum code);
00711 } // namespace Color
00712
00713 namespace assertType {
00714     enum Enum
00715     {
00716         // macro traits
00717
00718         is_warn    = 1,
00719         is_check   = 2 * is_warn,
00720         is_require = 2 * is_check,
00721

```

```

00722         is_normal      = 2 * is_require,
00723         is_throws      = 2 * is_normal,
00724         is_throws_as   = 2 * is_throws,
00725         is_throws_with = 2 * is_throws_as,
00726         is_nothrow     = 2 * is_throws_with,
00727
00728         is_false = 2 * is_nothrow,
00729         is_unary = 2 * is_false, // not checked anywhere - used just to distinguish the types
00730
00731         is_eq = 2 * is_unary,
00732         is_ne = 2 * is_eq,
00733
00734         is_lt = 2 * is_ne,
00735         is_gt = 2 * is_lt,
00736
00737         is_ge = 2 * is_gt,
00738         is_le = 2 * is_ge,
00739
00740         // macro types
00741
00742         DT_WARN      = is_normal | is_warn,
00743         DT_CHECK     = is_normal | is_check,
00744         DT_REQUIRE   = is_normal | is_require,
00745
00746         DT_WARN_FALSE = is_normal | is_false | is_warn,
00747         DT_CHECK_FALSE = is_normal | is_false | is_check,
00748         DT_REQUIRE_FALSE = is_normal | is_false | is_require,
00749
00750         DT_WARN_THROWS = is_throws | is_warn,
00751         DT_CHECK_THROWS = is_throws | is_check,
00752         DT_REQUIRE_THROWS = is_throws | is_require,
00753
00754         DT_WARN_THROWS_AS = is_throws_as | is_warn,
00755         DT_CHECK_THROWS_AS = is_throws_as | is_check,
00756         DT_REQUIRE_THROWS_AS = is_throws_as | is_require,
00757
00758         DT_WARN_THROWS_WITH = is_throws_with | is_warn,
00759         DT_CHECK_THROWS_WITH = is_throws_with | is_check,
00760         DT_REQUIRE_THROWS_WITH = is_throws_with | is_require,
00761
00762         DT_WARN_THROWS_WITH_AS = is_throws_with | is_throws_as | is_warn,
00763         DT_CHECK_THROWS_WITH_AS = is_throws_with | is_throws_as | is_check,
00764         DT_REQUIRE_THROWS_WITH_AS = is_throws_with | is_throws_as | is_require,
00765
00766         DT_WARN_NOTHROW = is_nothrow | is_warn,
00767         DT_CHECK_NOTHROW = is_nothrow | is_check,
00768         DT_REQUIRE_NOTHROW = is_nothrow | is_require,
00769
00770         DT_WARN_EQ = is_normal | is_eq | is_warn,
00771         DT_CHECK_EQ = is_normal | is_eq | is_check,
00772         DT_REQUIRE_EQ = is_normal | is_eq | is_require,
00773
00774         DT_WARN_NE = is_normal | is_ne | is_warn,
00775         DT_CHECK_NE = is_normal | is_ne | is_check,
00776         DT_REQUIRE_NE = is_normal | is_ne | is_require,
00777
00778         DT_WARN_GT = is_normal | is_gt | is_warn,
00779         DT_CHECK_GT = is_normal | is_gt | is_check,
00780         DT_REQUIRE_GT = is_normal | is_gt | is_require,
00781
00782         DT_WARN_LT = is_normal | is_lt | is_warn,
00783         DT_CHECK_LT = is_normal | is_lt | is_check,
00784         DT_REQUIRE_LT = is_normal | is_lt | is_require,
00785
00786         DT_WARN_GE = is_normal | is_ge | is_warn,
00787         DT_CHECK_GE = is_normal | is_ge | is_check,
00788         DT_REQUIRE_GE = is_normal | is_ge | is_require,
00789
00790         DT_WARN_LE = is_normal | is_le | is_warn,
00791         DT_CHECK_LE = is_normal | is_le | is_check,
00792         DT_REQUIRE_LE = is_normal | is_le | is_require,
00793
00794         DT_WARN_UNARY = is_normal | is_unary | is_warn,
00795         DT_CHECK_UNARY = is_normal | is_unary | is_check,
00796         DT_REQUIRE_UNARY = is_normal | is_unary | is_require,
00797
00798         DT_WARN_UNARY_FALSE = is_normal | is_false | is_unary | is_warn,
00799         DT_CHECK_UNARY_FALSE = is_normal | is_false | is_unary | is_check,
00800         DT_REQUIRE_UNARY_FALSE = is_normal | is_false | is_unary | is_require,
00801     };
00802 } // namespace assertType
00803
00804 DOCTEST_INTERFACE const char* assertString(assertType::Enum at);
00805 DOCTEST_INTERFACE const char* failureString(assertType::Enum at);
00806 DOCTEST_INTERFACE const char* skipPathFromFilename(const char* file);
00807
00808 struct DOCTEST_INTERFACE TestCaseData

```

```

00809 {
00810     String      m_file;           // the file in which the test was registered (using String - see #350)
00811     unsigned    m_line;           // the line where the test was registered
00812     const char* m_name;           // name of the test case
00813     const char* m_test_suite;     // the test suite in which the test was added
00814     const char* m_description;
00815     bool        m_skip;
00816     bool        m_no_breaks;
00817     bool        m_no_output;
00818     bool        m_may_fail;
00819     bool        m_should_fail;
00820     int         m_expected_failures;
00821     double      m_timeout;
00822 };
00823
00824 struct DOCTEST_INTERFACE AssertData
00825 {
00826     // common - for all asserts
00827     const TestCaseData* m_test_case;
00828     assertType::Enum    m_at;
00829     const char*         m_file;
00830     int                 m_line;
00831     const char*         m_expr;
00832     bool                m_failed;
00833
00834     // exception-related - for all asserts
00835     bool m_threw;
00836     String m_exception;
00837
00838     // for normal asserts
00839     String m_decomp;
00840
00841     // for specific exception-related asserts
00842     bool m_threw_as;
00843     const char* m_exception_type;
00844
00845     class DOCTEST_INTERFACE StringContains {
00846     private:
00847         Contains content;
00848         bool isContains;
00849
00850     public:
00851         StringContains(const String& str) : content(str), isContains(false) { }
00852         StringContains(Contains cntn) : content(static_cast<Contains&&>(cntn)), isContains(true) { }
00853     }
00854
00855     bool check(const String& str) { return isContains ? (content == str) : (content.string ==
str); }
00856
00857     operator const String&() const { return content.string; }
00858
00859     const char* c_str() const { return content.string.c_str(); }
00860 } m_exception_string;
00861
00862 AssertData(assertType::Enum at, const char* file, int line, const char* expr,
const char* exception_type, const StringContains& exception_string);
00863 };
00864
00865 struct DOCTEST_INTERFACE MessageData
00866 {
00867     String      m_string;
00868     const char* m_file;
00869     int         m_line;
00870     assertType::Enum m_severity;
00871 };
00872
00873 struct DOCTEST_INTERFACE SubcaseSignature
00874 {
00875     String      m_name;
00876     const char* m_file;
00877     int         m_line;
00878
00879     bool operator==(const SubcaseSignature& other) const;
00880     bool operator<(const SubcaseSignature& other) const;
00881 };
00882
00883 struct DOCTEST_INTERFACE IContextScope
00884 {
00885     DOCTEST_DECLARE_INTERFACE(IContextScope)
00886     virtual void stringify(std::ostream*) const = 0;
00887 };
00888
00889 namespace detail {
00890     struct DOCTEST_INTERFACE TestCase;
00891 } // namespace detail
00892
00893 struct ContextOptions

```

```

00894 {
00895     std::ostream* cout = nullptr; // stdout stream
00896     String      binary_name;      // the test binary name
00897
00898     const detail::TestCase* currentTest = nullptr;
00899
00900     // == parameters from the command line
00901     String out; // output filename
00902     String order_by; // how tests should be ordered
00903     unsigned rand_seed; // the seed for rand ordering
00904
00905     unsigned first; // the first (matching) test to be executed
00906     unsigned last; // the last (matching) test to be executed
00907
00908     int abort_after; // stop tests after this many failed assertions
00909     int subcase_filter_levels; // apply the subcase filters for the first N levels
00910
00911     bool success; // include successful assertions in output
00912     bool case_sensitive; // if filtering should be case sensitive
00913     bool exit; // if the program should be exited after the tests are ran/whatever
00914     bool duration; // print the time duration of each test case
00915     bool minimal; // minimal console output (only test failures)
00916     bool quiet; // no console output
00917     bool no_throw; // to skip exceptions-related assertion macros
00918     bool no_exitcode; // if the framework should return 0 as the exitcode
00919     bool no_run; // to not run the tests at all (can be done with an "*" exclude)
00920     bool no_intro; // to not print the intro of the framework
00921     bool no_version; // to not print the version of the framework
00922     bool no_colors; // if output to the console should be colorized
00923     bool force_colors; // forces the use of colors even when a tty cannot be detected
00924     bool no_breaks; // to not break into the debugger
00925     bool no_skip; // don't skip test cases which are marked to be skipped
00926     bool gnu_file_line; // if line numbers should be surrounded with :x: and not (x):
00927     bool no_path_in_filenames; // if the path to files should be removed from the output
00928     String strip_file_prefixes; // remove the longest matching one of these prefixes from any file
00929     paths in the output
00930     bool no_line_numbers; // if source code line numbers should be omitted from the output
00931     bool no_debug_output; // no output in the debug console when a debugger is attached
00932     bool no_skipped_summary; // don't print "skipped" in the summary !!! UNDOCUMENTED !!!
00933     bool no_time_in_output; // omit any time/timestamps from output !!! UNDOCUMENTED !!!
00934
00935     bool help; // to print the help
00936     bool version; // to print the version
00937     bool count; // if only the count of matching tests is to be retrieved
00938     bool list_test_cases; // to list all tests matching the filters
00939     bool list_test_suites; // to list all suites matching the filters
00940     bool list_reporters; // lists all registered reporters
00941 };
00942 namespace detail {
00943     namespace types {
00944         #ifdef DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
00945             using namespace std;
00946         #else
00947             template <bool COND, typename T = void>
00948             struct enable_if { };
00949
00950             template <typename T>
00951             struct enable_if<true, T> { using type = T; };
00952
00953             struct true_type { static DOCTEST_CONSTEXPR bool value = true; };
00954             struct false_type { static DOCTEST_CONSTEXPR bool value = false; };
00955
00956             template <typename T> struct remove_reference { using type = T; };
00957             template <typename T> struct remove_reference<T&> { using type = T; };
00958             template <typename T> struct remove_reference<T&&> { using type = T; };
00959
00960             template <typename T> struct is_rvalue_reference : false_type { };
00961             template <typename T> struct is_rvalue_reference<T&&> : true_type { };
00962
00963             template <typename T> struct remove_const { using type = T; };
00964             template <typename T> struct remove_const<const T> { using type = T; };
00965
00966             // Compiler intrinsics
00967             template <typename T> struct is_enum { static DOCTEST_CONSTEXPR bool value = __is_enum(T); };
00968             template <typename T> struct underlying_type { using type = __underlying_type(T); };
00969
00970             template <typename T> struct is_pointer : false_type { };
00971             template <typename T> struct is_pointer<T*> : true_type { };
00972
00973             template <typename T> struct is_array : false_type { };
00974             // NOLINTNEXTLINE(*-avoid-c-arrays)
00975             template <typename T, size_t SIZE> struct is_array<T[SIZE]> : true_type { };
00976         #endif
00977     }
00978
00979     // <utility>

```

```

00980     template <typename T>
00981     T&& declval();
00982
00983     template <class T>
00984     DOCTEST_CONSTEXPR_FUNC T&& forward(typename types::remove_reference<T>::type& t) DOCTEST_NOEXCEPT
00985     {
00986         return static_cast<T&&>(t);
00987     }
00988
00989     template <class T>
00990     DOCTEST_CONSTEXPR_FUNC T&& forward(typename types::remove_reference<T>::type&& t) DOCTEST_NOEXCEPT
00991     {
00992         return static_cast<T&&>(t);
00993     }
00994
00995     template <typename T>
00996     struct deferred_false : types::false_type { };
00997
00998     // MSVS 2015 :(
00999     #if !DOCTEST_CLANG && defined(_MSC_VER) && _MSC_VER <= 1900
01000     template <typename T, typename = void>
01001     struct has_global_insertion_operator : types::false_type { };
01002
01003     template <typename T>
01004     struct has_global_insertion_operator<T, decltype(::operator<<(declval<std::ostream&>()), declval<const T&>()), void())> : types::true_type { };
01005
01006     template <typename T, typename = void>
01007     struct has_insertion_operator { static DOCTEST_CONSTEXPR bool value =
01008     has_global_insertion_operator<T>::value; };
01009
01010     template <typename T, bool global>
01011     struct insert_hack;
01012
01013     template <typename T>
01014     struct insert_hack<T, true> {
01015         static void insert(std::ostream& os, const T& t) { ::operator<<(os, t); }
01016     };
01017
01018     template <typename T>
01019     struct insert_hack<T, false> {
01020         static void insert(std::ostream& os, const T& t) { operator<<(os, t); }
01021     };
01022
01023     template <typename T>
01024     using insert_hack_t = insert_hack<T, has_global_insertion_operator<T>::value>;
01025
01026     #else
01027     template <typename T, typename = void>
01028     struct has_insertion_operator : types::false_type { };
01029
01030     #endif
01031
01032     template <typename T>
01033     struct should_stringify_as_underlying_type {
01034         static DOCTEST_CONSTEXPR bool value = detail::types::is_enum<T>::value &&
01035         !doctest::detail::has_insertion_operator<T>::value;
01036     };
01037
01038     DOCTEST_INTERFACE std::ostream* tlssPush();
01039     DOCTEST_INTERFACE String tlssPop();
01040
01041     template <bool C>
01042     struct StringMakerBase {
01043         template <typename T>
01044         static String convert(const DOCTEST_REF_WRAP(T)) {
01045             #ifdef DOCTEST_CONFIG_REQUIRE_STRINGIFICATION_FOR_ALL_USED_TYPES
01046             static_assert(deferred_false<T>::value, "No stringification detected for type T. See
01047             string conversion manual");
01048             #endif
01049             return "{?}";
01050         }
01051     };
01052
01053     template <typename T>
01054     struct filldata;
01055
01056     template <typename T>
01057     void fillloss(std::ostream* stream, const T& in) {
01058         filldata<T>::fill(stream, in);
01059     }
01060
01061     template <typename T, size_t N>
01062     void fillloss(std::ostream* stream, const T (&in)[N]) { // NOLINT(*-avoid-c-arrays)
01063         // T[N], T(&)[N], T(&&)[N] have same behaviour.

```

```

01060         // Hence remove reference.
01061         filloss<typename> types::remove_reference<decltype(in)>::type>(stream, in);
01062     }
01063
01064     template <typename T>
01065     String toStream(const T& in) {
01066         std::ostream* stream = tlssPush();
01067         filloss(stream, in);
01068         return tlssPop();
01069     }
01070
01071     template <>
01072     struct StringMakerBase<true> {
01073         template <typename T>
01074         static String convert(const DOCTEST_REF_WRAP(T) in) {
01075             return toStream(in);
01076         }
01077     };
01078 } // namespace detail
01079
01080 template <typename T>
01081 struct StringMaker : public detail::StringMakerBase<
01082     detail::has_insertion_operator<T>::value || detail::types::is_pointer<T>::value ||
01083     detail::types::is_array<T>::value>
01084 {};
01085
01086 #ifndef DOCTEST_STRINGIFY
01087 #define DOCTEST_CONFIG_DOUBLE_STRINGIFY
01088 #define DOCTEST_STRINGIFY(...) toString(toString(__VA_ARGS__))
01089 #else
01090 #define DOCTEST_STRINGIFY(...) toString(__VA_ARGS__)
01091 #endif
01092
01093 template <typename T>
01094 String toString() {
01095     #if DOCTEST_CLANG == 0 && DOCTEST_GCC == 0 && DOCTEST_ICC == 0
01096         String ret = __FUNCSIG__; // class doctest::String __cdecl doctest::toString<TYPE>(void)
01097         String::size_type beginPos = ret.find('<');
01098         return ret.substr(beginPos + 1, ret.size() - beginPos -
01099             static_cast<String::size_type>(sizeof(">(void)")));
01100     #else
01101         String ret = __PRETTY_FUNCTION__; // doctest::String toString() [with T = TYPE]
01102         String::size_type begin = ret.find('=') + 2;
01103         return ret.substr(begin, ret.size() - begin - 1);
01104     #endif
01105 }
01106
01107 template <typename T, typename
01108     detail::types::enable_if<!detail::should_stringify_as_underlying_type<T>::value, bool>::type = true>
01109 String toString(const DOCTEST_REF_WRAP(T) value) {
01110     return StringMaker<T>::convert(value);
01111 }
01112
01113 #ifdef DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
01114 DOCTEST_INTERFACE String toString(const char* in);
01115 #endif // DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
01116
01117 #if DOCTEST_MSVC >= DOCTEST_COMPILER(19, 20, 0)
01118 // see this issue on why this is needed: https://github.com/doctest/doctest/issues/183
01119 DOCTEST_INTERFACE String toString(const std::string& in);
01120 #endif // VS 2019
01121
01122 DOCTEST_INTERFACE String toString(String in);
01123
01124 DOCTEST_INTERFACE String toString(std::nullptr_t);
01125
01126 DOCTEST_INTERFACE String toString(bool in);
01127
01128 DOCTEST_INTERFACE String toString(float in);
01129 DOCTEST_INTERFACE String toString(double in);
01130 DOCTEST_INTERFACE String toString(double long in);
01131
01132 DOCTEST_INTERFACE String toString(char in);
01133 DOCTEST_INTERFACE String toString(char signed in);
01134 DOCTEST_INTERFACE String toString(char unsigned in);
01135 DOCTEST_INTERFACE String toString(short in);
01136 DOCTEST_INTERFACE String toString(short unsigned in);
01137 DOCTEST_INTERFACE String toString(signed in);
01138 DOCTEST_INTERFACE String toString(unsigned in);
01139 DOCTEST_INTERFACE String toString(long in);
01140 DOCTEST_INTERFACE String toString(long unsigned in);
01141 DOCTEST_INTERFACE String toString(long long in);
01142 DOCTEST_INTERFACE String toString(long long unsigned in);
01143
01144 template <typename T, typename
01145     detail::types::enable_if<detail::should_stringify_as_underlying_type<T>::value, bool>::type = true>

```

```

01143 String toString(const DOCTEST_REF_WRAP(T) value) {
01144     using UT = typename detail::types::underlying_type<T>::type;
01145     return (DOCTEST_STRINGIFY(static_cast<UT>(value)));
01146 }
01147
01148 namespace detail {
01149     template <typename T>
01150     struct filldata
01151     {
01152         static void fill(std::ostream* stream, const T& in) {
01153             #if defined(_MSC_VER) && _MSC_VER <= 1900
01154                 insert_hack_t<T>::insert(*stream, in);
01155             #else
01156                 operator<<(*stream, in);
01157             #endif
01158         }
01159     };
01160
01161     DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4866)
01162     // NOLINTBEGIN(*-avoid-c-arrays)
01163     template <typename T, size_t N>
01164     struct filldata<T[N]> {
01165         static void fill(std::ostream* stream, const T(&in)[N]) {
01166             *stream << "[";
01167             for (size_t i = 0; i < N; i++) {
01168                 if (i != 0) { *stream << ", "; }
01169                 *stream << (DOCTEST_STRINGIFY(in[i]));
01170             }
01171             *stream << "];";
01172         }
01173     };
01174     // NOLINTEND(*-avoid-c-arrays)
01175     DOCTEST_MSVC_SUPPRESS_WARNING_POP
01176
01177     // Specialized since we don't want the terminating null byte!
01178     // NOLINTBEGIN(*-avoid-c-arrays)
01179     template <size_t N>
01180     struct filldata<const char[N]> {
01181         static void fill(std::ostream* stream, const char (&in)[N]) {
01182             *stream << String(in, in[N - 1] ? N : N - 1);
01183         } // NOLINT(clang-analyzer-cplusplus.NewDeleteLeaks)
01184     };
01185     // NOLINTEND(*-avoid-c-arrays)
01186
01187     template <>
01188     struct filldata<const void*> {
01189         static void fill(std::ostream* stream, const void* in);
01190     };
01191
01192     template <typename T>
01193     struct filldata<T*> {
01194         DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4180)
01195         static void fill(std::ostream* stream, const T* in) {
01196             DOCTEST_MSVC_SUPPRESS_WARNING_POP
01197             DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wmicrosoft-cast")
01198             filldata<const void*>::fill(stream,
01199             #if DOCTEST_GCC == 0 || DOCTEST_GCC >= DOCTEST_COMPILER(4, 9, 0)
01200                 reinterpret_cast<const void*>(in)
01201             #else
01202                 *reinterpret_cast<const void* const*>(&in)
01203             #endif
01204             );
01205             DOCTEST_CLANG_SUPPRESS_WARNING_POP
01206         }
01207     };
01208 }
01209
01210 struct DOCTEST_INTERFACE Approx
01211 {
01212     Approx(double value);
01213
01214     Approx operator()(double value) const;
01215
01216     #ifndef DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
01217     template <typename T>
01218     explicit Approx(const T& value,
01219         typename detail::types::enable_if<std::is_constructible<double, T>::value>::type*
01220         =
01221         static_cast<T*>(nullptr)) {
01222         *this = static_cast<double>(value);
01223     }
01224     #endif // DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
01225
01226     Approx& epsilon(double newEpsilon);
01227     #ifndef DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
01228     template <typename T>

```



```

01229     typename std::enable_if<std::is_constructible<double, T>::value, Approx&>::type epsilon(
01230         const T& newEpsilon) {
01231         m_epsilon = static_cast<double>(newEpsilon);
01232         return *this;
01233     }
01234 #endif // DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
01235
01236     Approx& scale(double newScale);
01237
01238 #ifdef DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
01239     template <typename T>
01240     typename std::enable_if<std::is_constructible<double, T>::value, Approx&>::type scale(
01241         const T& newScale) {
01242         m_scale = static_cast<double>(newScale);
01243         return *this;
01244     }
01245 #endif // DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
01246
01247     // clang-format off
01248     DOCTEST_INTERFACE friend bool operator==(double lhs, const Approx& rhs);
01249     DOCTEST_INTERFACE friend bool operator==(const Approx& lhs, double rhs);
01250     DOCTEST_INTERFACE friend bool operator!=(double lhs, const Approx& rhs);
01251     DOCTEST_INTERFACE friend bool operator!=(const Approx& lhs, double rhs);
01252     DOCTEST_INTERFACE friend bool operator<=(double lhs, const Approx& rhs);
01253     DOCTEST_INTERFACE friend bool operator<=(const Approx& lhs, double rhs);
01254     DOCTEST_INTERFACE friend bool operator>=(double lhs, const Approx& rhs);
01255     DOCTEST_INTERFACE friend bool operator>=(const Approx& lhs, double rhs);
01256     DOCTEST_INTERFACE friend bool operator< (double lhs, const Approx& rhs);
01257     DOCTEST_INTERFACE friend bool operator< (const Approx& lhs, double rhs);
01258     DOCTEST_INTERFACE friend bool operator> (double lhs, const Approx& rhs);
01259     DOCTEST_INTERFACE friend bool operator> (const Approx& lhs, double rhs);
01260
01261 #ifdef DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
01262 #define DOCTEST_APPROX_PREFIX \
01263     template <typename T> friend typename std::enable_if<std::is_constructible<double, T>::value,
01264     bool>::type
01265     DOCTEST_APPROX_PREFIX operator==(const T& lhs, const Approx& rhs) { return
01266     operator==(static_cast<double>(lhs), rhs); }
01267     DOCTEST_APPROX_PREFIX operator==(const Approx& lhs, const T& rhs) { return operator==(rhs, lhs); }
01268     DOCTEST_APPROX_PREFIX operator!=(const T& lhs, const Approx& rhs) { return !operator==(lhs, rhs); }
01269     DOCTEST_APPROX_PREFIX operator!=(const Approx& lhs, const T& rhs) { return !operator==(rhs, lhs); }
01270     DOCTEST_APPROX_PREFIX operator<=(const T& lhs, const Approx& rhs) { return
01271     static_cast<double>(lhs) < rhs.m_value || lhs == rhs; }
01272     DOCTEST_APPROX_PREFIX operator<=(const Approx& lhs, const T& rhs) { return lhs.m_value <
01273     static_cast<double>(rhs) || lhs == rhs; }
01274     DOCTEST_APPROX_PREFIX operator>=(const T& lhs, const Approx& rhs) { return
01275     static_cast<double>(lhs) > rhs.m_value || lhs == rhs; }
01276     DOCTEST_APPROX_PREFIX operator>=(const Approx& lhs, const T& rhs) { return lhs.m_value >
01277     static_cast<double>(rhs) || lhs == rhs; }
01278     DOCTEST_APPROX_PREFIX operator< (const T& lhs, const Approx& rhs) { return
01279     static_cast<double>(lhs) < rhs.m_value && lhs != rhs; }
01280     DOCTEST_APPROX_PREFIX operator< (const Approx& lhs, const T& rhs) { return lhs.m_value <
01281     static_cast<double>(rhs) && lhs != rhs; }
01282     DOCTEST_APPROX_PREFIX operator> (const T& lhs, const Approx& rhs) { return
01283     static_cast<double>(lhs) > rhs.m_value && lhs != rhs; }
01284     DOCTEST_APPROX_PREFIX operator> (const Approx& lhs, const T& rhs) { return lhs.m_value >
01285     static_cast<double>(rhs) && lhs != rhs; }
01286 #undef DOCTEST_APPROX_PREFIX
01287 #endif // DOCTEST_CONFIG_INCLUDE_TYPE_TRAITS
01288
01289     // clang-format on
01290
01291     double m_epsilon;
01292     double m_scale;
01293     double m_value;
01294 };
01295
01296 DOCTEST_INTERFACE String toString(const Approx& in);
01297
01298 DOCTEST_INTERFACE const ContextOptions* getContextOptions();
01299
01300 template <typename F>
01301 struct DOCTEST_INTERFACE_DECL IsNaN
01302 {
01303     F value; bool flipped;
01304     IsNaN(F f, bool flip = false) : value(f), flipped(flip) { }
01305     IsNaN<F> operator!() const { return { value, !flipped }; }
01306     operator bool() const;
01307 };
01308
01309 #ifndef __MINGW32__
01310 extern template struct DOCTEST_INTERFACE_DECL IsNaN<float>;
01311 extern template struct DOCTEST_INTERFACE_DECL IsNaN<double>;
01312 extern template struct DOCTEST_INTERFACE_DECL IsNaN<long double>;
01313 #endif

```

```

01304 DOCTEST_INTERFACE String toString(IsNaN<float> in);
01305 DOCTEST_INTERFACE String toString(IsNaN<double> in);
01306 DOCTEST_INTERFACE String toString(IsNaN<double long> in);
01307
01308 #ifndef DOCTEST_CONFIG_DISABLE
01309
01310 namespace detail {
01311     // clang-format off
01312     #ifndef DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
01313         template<class T> struct decay_array { using type = T; };
01314         template<class T, unsigned N> struct decay_array<T[N]> { using type = T*; };
01315         template<class T> struct decay_array<T[]> { using type = T*; };
01316
01317         template<class T> struct not_char_pointer { static DOCTEST_CONSTEXPR int value = 1; };
01318         template<> struct not_char_pointer<char*> { static DOCTEST_CONSTEXPR int value = 0; };
01319         template<> struct not_char_pointer<const char*> { static DOCTEST_CONSTEXPR int value = 0; };
01320     };
01321     template<class T> struct can_use_op : public not_char_pointer<typename decay_array<T>::type> {};
01322 #endif // DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
01323     // clang-format on
01324
01325     struct DOCTEST_INTERFACE TestFailureException
01326     {
01327     };
01328
01329     DOCTEST_INTERFACE bool checkIfShouldThrow(assertType::Enum at);
01330
01331 #ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
01332     DOCTEST_NORETURN
01333 #endif // DOCTEST_CONFIG_NO_EXCEPTIONS
01334     DOCTEST_INTERFACE void throwException();
01335
01336     struct DOCTEST_INTERFACE Subcase
01337     {
01338         SubcaseSignature m_signature;
01339         bool m_entered = false;
01340
01341         Subcase(const String& name, const char* file, int line);
01342         Subcase(const Subcase&) = delete;
01343         Subcase(Subcase&&) = delete;
01344         Subcase& operator=(const Subcase&) = delete;
01345         Subcase& operator=(Subcase&&) = delete;
01346         ~Subcase();
01347
01348         operator bool() const;
01349
01350     private:
01351         bool checkFilters();
01352     };
01353
01354     template <typename L, typename R>
01355     String stringifyBinaryExpr(const DOCTEST_REF_WRAP(L) lhs, const char* op,
01356                               const DOCTEST_REF_WRAP(R) rhs) {
01357         return (DOCTEST_STRINGIFY(lhs)) + op + (DOCTEST_STRINGIFY(rhs));
01358     }
01359
01360 #if DOCTEST_CLANG && DOCTEST_CLANG < DOCTEST_COMPILER(3, 6, 0)
01361     DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wunused-comparison")
01362 #endif
01363
01364 // This will check if there is any way it could find a operator like member or friend and uses it.
01365 // If not it doesn't find the operator or if the operator at global scope is defined after
01366 // this template, the template won't be instantiated due to SFINAE. Once the template is not
01367 // instantiated it can look for global operator using normal conversions.
01368 #ifndef __NVCC__
01369 #define SFINAE_OP(ret,op) ret
01370 #else
01371 #define SFINAE_OP(ret,op) decltype((void)(doctest::detail::declval<L>() op
doctest::detail::declval<R>()),ret{})
01372 #endif
01373
01374 #define DOCTEST_DO_BINARY_EXPRESSION_COMPARISON(op, op_str, op_macro)
01375     template <typename R>
01376     DOCTEST_NOINLINE SFINAE_OP(Result,op) operator op(R&& rhs) {
01377         bool res = op_macro(doctest::detail::forward<const L>(lhs), doctest::detail::forward<R>(rhs));
01378         if(m_at & assertType::is_false)
01379             res = !res;
01380         if(!res || doctest::getContextOptions()->success)
01381             return Result(res, stringifyBinaryExpr(lhs, op_str, rhs));
01382         return Result(res);
01383     }
01384
01385 // more checks could be added - like in Catch:
01386 // https://github.com/catchorg/Catch2/pull/1480/files

```

```

01387 // https://github.com/catchorg/Catch2/pull/1481/files
01388 #define DOCTEST_FORBIT_EXPRESSION(rt, op)
01389     template <typename R>
01390     rt& operator op(const R&) {
01391         static_assert(deferred_false<R>::value,
01392             "Expression Too Complex Please Rewrite As Binary Comparison!");
01393     }
01394 }
01395
01396 struct DOCTEST_INTERFACE Result // NOLINT(*-member-init)
01397 {
01398     bool m_passed;
01399     String m_decomp;
01400
01401     Result() = default; // TODO: Why do we need this? (To remove NOLINT)
01402     Result(bool passed, const String& decomposition = String());
01403
01404     // forbidding some expressions based on this table:
01405     // https://en.cppreference.com/w/cpp/language/operator_precedence
01406     DOCTEST_FORBIT_EXPRESSION(Result, &)
01407     DOCTEST_FORBIT_EXPRESSION(Result, ^)
01408     DOCTEST_FORBIT_EXPRESSION(Result, |)
01409     DOCTEST_FORBIT_EXPRESSION(Result, &&)
01410     DOCTEST_FORBIT_EXPRESSION(Result, ||)
01411     DOCTEST_FORBIT_EXPRESSION(Result, ==)
01412     DOCTEST_FORBIT_EXPRESSION(Result, !=)
01413     DOCTEST_FORBIT_EXPRESSION(Result, <)
01414     DOCTEST_FORBIT_EXPRESSION(Result, >)
01415     DOCTEST_FORBIT_EXPRESSION(Result, <=)
01416     DOCTEST_FORBIT_EXPRESSION(Result, >=)
01417     DOCTEST_FORBIT_EXPRESSION(Result, =)
01418     DOCTEST_FORBIT_EXPRESSION(Result, +=)
01419     DOCTEST_FORBIT_EXPRESSION(Result, -=)
01420     DOCTEST_FORBIT_EXPRESSION(Result, *=)
01421     DOCTEST_FORBIT_EXPRESSION(Result, /=)
01422     DOCTEST_FORBIT_EXPRESSION(Result, %=)
01423     DOCTEST_FORBIT_EXPRESSION(Result, <<=)
01424     DOCTEST_FORBIT_EXPRESSION(Result, >>=)
01425     DOCTEST_FORBIT_EXPRESSION(Result, &=)
01426     DOCTEST_FORBIT_EXPRESSION(Result, ^=)
01427     DOCTEST_FORBIT_EXPRESSION(Result, |=)
01428 };
01429 #ifndef DOCTEST_CONFIG_NO_COMPARISON_WARNING_SUPPRESSION
01430
01431     DOCTEST_CLANG_SUPPRESS_WARNING_PUSH
01432     DOCTEST_CLANG_SUPPRESS_WARNING("-Wsign-conversion")
01433     DOCTEST_CLANG_SUPPRESS_WARNING("-Wsign-compare")
01434     //DOCTEST_CLANG_SUPPRESS_WARNING("-Wdouble-promotion")
01435     //DOCTEST_CLANG_SUPPRESS_WARNING("-Wconversion")
01436     //DOCTEST_CLANG_SUPPRESS_WARNING("-Wfloat-equal")
01437
01438     DOCTEST_GCC_SUPPRESS_WARNING_PUSH
01439     DOCTEST_GCC_SUPPRESS_WARNING("-Wsign-conversion")
01440     DOCTEST_GCC_SUPPRESS_WARNING("-Wsign-compare")
01441     //DOCTEST_GCC_SUPPRESS_WARNING("-Wdouble-promotion")
01442     //DOCTEST_GCC_SUPPRESS_WARNING("-Wconversion")
01443     //DOCTEST_GCC_SUPPRESS_WARNING("-Wfloat-equal")
01444
01445     DOCTEST_MSVC_SUPPRESS_WARNING_PUSH
01446     // https://stackoverflow.com/questions/39479163 what's the difference between 4018 and 4389
01447     DOCTEST_MSVC_SUPPRESS_WARNING(4388) // signed/unsigned mismatch
01448     DOCTEST_MSVC_SUPPRESS_WARNING(4389) // 'operator' : signed/unsigned mismatch
01449     DOCTEST_MSVC_SUPPRESS_WARNING(4018) // 'expression' : signed/unsigned mismatch
01450     //DOCTEST_MSVC_SUPPRESS_WARNING(4805) // 'operation' : unsafe mix of type 'type' and type 'type'
01451     in operation
01452 #endif // DOCTEST_CONFIG_NO_COMPARISON_WARNING_SUPPRESSION
01453
01454 // clang-format off
01455 #ifndef DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
01456 #define DOCTEST_COMPARISON_RETURN_TYPE bool
01457 #else // DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
01458 #define DOCTEST_COMPARISON_RETURN_TYPE typename types::enable_if<can_use_op<L>::value ||
01459     can_use_op<R>::value, bool>::type
01460
01461     inline bool eq(const char* lhs, const char* rhs) { return String(lhs) == String(rhs); }
01462     inline bool ne(const char* lhs, const char* rhs) { return String(lhs) != String(rhs); }
01463     inline bool lt(const char* lhs, const char* rhs) { return String(lhs) < String(rhs); }
01464     inline bool gt(const char* lhs, const char* rhs) { return String(lhs) > String(rhs); }
01465     inline bool le(const char* lhs, const char* rhs) { return String(lhs) <= String(rhs); }
01466     inline bool ge(const char* lhs, const char* rhs) { return String(lhs) >= String(rhs); }
01467 #endif // DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
01468 // clang-format on
01469
01470 #define DOCTEST_RELATIONAL_OP(name, op)
01471     template <typename L, typename R>
01472     DOCTEST_COMPARISON_RETURN_TYPE name(const DOCTEST_REF_WRAP(L) lhs,

```

```

01471                                     const DOCTEST_REF_WRAP(R) rhs) {
01472         return lhs op rhs;
01473     }
01474
01475     DOCTEST_RELATIONAL_OP(eq, ==)
01476     DOCTEST_RELATIONAL_OP(ne, !=)
01477     DOCTEST_RELATIONAL_OP(lt, <)
01478     DOCTEST_RELATIONAL_OP(gt, >)
01479     DOCTEST_RELATIONAL_OP(le, <=)
01480     DOCTEST_RELATIONAL_OP(ge, >=)
01481
01482 #ifndef DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
01483 #define DOCTEST_CMP_EQ(l, r) l == r
01484 #define DOCTEST_CMP_NE(l, r) l != r
01485 #define DOCTEST_CMP_GT(l, r) l > r
01486 #define DOCTEST_CMP_LT(l, r) l < r
01487 #define DOCTEST_CMP_GE(l, r) l >= r
01488 #define DOCTEST_CMP_LE(l, r) l <= r
01489 #else // DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
01490 #define DOCTEST_CMP_EQ(l, r) eq(l, r)
01491 #define DOCTEST_CMP_NE(l, r) ne(l, r)
01492 #define DOCTEST_CMP_GT(l, r) gt(l, r)
01493 #define DOCTEST_CMP_LT(l, r) lt(l, r)
01494 #define DOCTEST_CMP_GE(l, r) ge(l, r)
01495 #define DOCTEST_CMP_LE(l, r) le(l, r)
01496 #endif // DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
01497
01498     template <typename L>
01499     // cppcheck-suppress copyCtorAndEqOperator
01500     struct Expression_lhs
01501     {
01502         L lhs;
01503         assertType::Enum m_at;
01504
01505         explicit Expression_lhs(L& in, assertType::Enum at)
01506             : lhs(static_cast<L&&>(in))
01507             , m_at(at) {}
01508
01509         DOCTEST_NOINLINE operator Result() {
01510             // this is needed only for MSVC 2015
01511             DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4800) // 'int': forcing value to bool
01512             bool res = static_cast<bool>(lhs);
01513             DOCTEST_MSVC_SUPPRESS_WARNING_POP
01514             if(m_at & assertType::is_false) {
01515                 res = !res;
01516             }
01517
01518             if(!res || getContextOptions()->success) {
01519                 return { res, (DOCTEST_STRINGIFY(lhs)) };
01520             }
01521             return { res };
01522         }
01523
01524         /* This is required for user-defined conversions from Expression_lhs to L */
01525         operator L() const { return lhs; }
01526
01527         // clang-format off
01528         DOCTEST_DO_BINARY_EXPRESSION_COMPARISON(==, " == ", DOCTEST_CMP_EQ)
01529         DOCTEST_DO_BINARY_EXPRESSION_COMPARISON(!=, " != ", DOCTEST_CMP_NE)
01530         DOCTEST_DO_BINARY_EXPRESSION_COMPARISON(>, " > ", DOCTEST_CMP_GT)
01531         DOCTEST_DO_BINARY_EXPRESSION_COMPARISON(<, " < ", DOCTEST_CMP_LT)
01532         DOCTEST_DO_BINARY_EXPRESSION_COMPARISON(>=, " >= ", DOCTEST_CMP_GE)
01533         DOCTEST_DO_BINARY_EXPRESSION_COMPARISON(<=, " <= ", DOCTEST_CMP_LE)
01534         // clang-format on
01535
01536         // forbidding some expressions based on this table:
01537         https://en.cppreference.com/w/cpp/language/operator\_precedence
01538         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, &)
01539         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, ^)
01540         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, |)
01541         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, &&)
01542         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, ||)
01543         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, =)
01544         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, +=)
01545         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, -=)
01546         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, *=)
01547         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, /=)
01548         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, %=)
01549         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, <<=)
01550         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, >>=)
01551         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, &=)
01552         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, ^=)
01553         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, |=)
01554         // these 2 are unfortunate because they should be allowed - they have higher precedence over
01555         // the comparisons, but the
01556         // ExpressionDecomposer class uses the left shift operator to capture the left operand of the
01557         // binary expression...

```

```

01555         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, «)
01556         DOCTEST_FORBIT_EXPRESSION(Expression_lhs, »)
01557     };
01558
01559 #ifndef DOCTEST_CONFIG_NO_COMPARISON_WARNING_SUPPRESSION
01560
01561     DOCTEST_CLANG_SUPPRESS_WARNING_POP
01562     DOCTEST_MSVC_SUPPRESS_WARNING_POP
01563     DOCTEST_GCC_SUPPRESS_WARNING_POP
01564
01565 #endif // DOCTEST_CONFIG_NO_COMPARISON_WARNING_SUPPRESSION
01566
01567 #if DOCTEST_CLANG && DOCTEST_CLANG < DOCTEST_COMPILER(3, 6, 0)
01568 DOCTEST_CLANG_SUPPRESS_WARNING_POP
01569 #endif
01570
01571     struct DOCTEST_INTERFACE ExpressionDecomposer
01572     {
01573         assertType::Enum m_at;
01574
01575         ExpressionDecomposer(assertType::Enum at);
01576
01577         // The right operator for capturing expressions is "<=" instead of "<<" (based on the operator
precedence table)
01578         // but then there will be warnings from GCC about "-Wparentheses" and since "_Pragma()" is
problematic this will stay for now...
01579         // https://github.com/catchorg/Catch2/issues/870
01580         // https://github.com/catchorg/Catch2/issues/565
01581         template <typename L>
01582         Expression_lhs<const L&&> operator<<(const L&& operand) { //bitfields bind to universal ref but
not const rvalue ref
01583             return Expression_lhs<const L&&>(static_cast<const L&&>(operand), m_at);
01584         }
01585
01586         template <typename L,typename
types::enable_if<!doctest::detail::types::is_rvalue_reference<L>::value,void >::type* = nullptr>
01587         Expression_lhs<const L&> operator<<(const L &operand) {
01588             return Expression_lhs<const L&>(operand, m_at);
01589         }
01590     };
01591
01592     struct DOCTEST_INTERFACE TestSuite
01593     {
01594         const char* m_test_suite = nullptr;
01595         const char* m_description = nullptr;
01596         bool m_skip = false;
01597         bool m_no_breaks = false;
01598         bool m_no_output = false;
01599         bool m_may_fail = false;
01600         bool m_should_fail = false;
01601         int m_expected_failures = 0;
01602         double m_timeout = 0;
01603
01604         TestSuite& operator*(const char* in);
01605
01606         template <typename T>
01607         TestSuite& operator*(const T& in) {
01608             in.fill(*this);
01609             return *this;
01610         }
01611     };
01612
01613     using funcType = void (*)();
01614
01615     struct DOCTEST_INTERFACE TestCase : public TestCaseData
01616     {
01617         funcType m_test; // a function pointer to the test case
01618
01619         String m_type; // for templated test cases - gets appended to the real name
01620         int m_template_id; // an ID used to distinguish between the different versions of a templated
test case
01621         String m_full_name; // contains the name (only for templated test cases!) + the template type
01622
01623         TestCase(funcType test, const char* file, unsigned line, const TestSuite& test_suite,
const String& type = String(), int template_id = -1);
01624
01625         TestCase(const TestCase& other);
01626         TestCase(TestCase&&) = delete;
01627
01628         DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(26434) // hides a non-virtual function
01629         TestCase& operator=(const TestCase& other);
01630         DOCTEST_MSVC_SUPPRESS_WARNING_POP
01631
01632         TestCase& operator=(TestCase&&) = delete;
01633
01634         TestCase& operator*(const char* in);
01635
01636

```

```

01637     template <typename T>
01638     TestCase& operator*(const T& in) {
01639         in.fill(*this);
01640         return *this;
01641     }
01642
01643     bool operator<(const TestCase& other) const;
01644
01645     ~TestCase() = default;
01646 };
01647
01648 // forward declarations of functions used by the macros
01649 DOCTEST_INTERFACE int regTest(const TestCase& tc);
01650 DOCTEST_INTERFACE int setTestSuite(const TestSuite& ts);
01651 DOCTEST_INTERFACE bool isDebuggerActive();
01652
01653 template<typename T>
01654 int instantiationHelper(const T&) { return 0; }
01655
01656 namespace binaryAssertComparison {
01657     enum Enum
01658     {
01659         eq = 0,
01660         ne,
01661         gt,
01662         lt,
01663         ge,
01664         le
01665     };
01666 } // namespace binaryAssertComparison
01667
01668 // clang-format off
01669 template <int, class L, class R> struct RelationalComparator { bool operator()(const
DOCTEST_REF_WRAP(L), const DOCTEST_REF_WRAP(R) ) const { return false; } };
01670
01671 #define DOCTEST_BINARY_RELATIONAL_OP(n, op) \
01672     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); } };
01673 // clang-format on
01674
01675 DOCTEST_BINARY_RELATIONAL_OP(0, doctest::detail::eq)
01676 DOCTEST_BINARY_RELATIONAL_OP(1, doctest::detail::ne)
01677 DOCTEST_BINARY_RELATIONAL_OP(2, doctest::detail::gt)
01678 DOCTEST_BINARY_RELATIONAL_OP(3, doctest::detail::lt)
01679 DOCTEST_BINARY_RELATIONAL_OP(4, doctest::detail::ge)
01680 DOCTEST_BINARY_RELATIONAL_OP(5, doctest::detail::le)
01681
01682 struct DOCTEST_INTERFACE ResultBuilder : public AssertData
01683 {
01684     ResultBuilder(assertType::Enum at, const char* file, int line, const char* expr,
const char* exception_type = "", const String& exception_string = "");
01685
01686     ResultBuilder(assertType::Enum at, const char* file, int line, const char* expr,
const char* exception_type, const Contains& exception_string);
01687
01688     void setResult(const Result& res);
01689
01690     template <int comparison, typename L, typename R>
DOCTEST_NOINLINE bool binary_assert(const DOCTEST_REF_WRAP(L) lhs,
const DOCTEST_REF_WRAP(R) rhs) {
01691         m_failed = !RelationalComparator<comparison, L, R>()(lhs, rhs);
01692         if (m_failed || getContextOptions()->success) {
01693             m_decomp = stringifyBinaryExpr(lhs, " ", rhs);
01694         }
01695         return !m_failed;
01696     }
01697
01698     template <typename L>
DOCTEST_NOINLINE bool unary_assert(const DOCTEST_REF_WRAP(L) val) {
01699         m_failed = !val;
01700
01701         if (m_at & assertType::is_false) {
01702             m_failed = !m_failed;
01703         }
01704
01705         if (m_failed || getContextOptions()->success) {
01706             m_decomp = (DOCTEST_STRINGIFY(val));
01707         }
01708
01709         return !m_failed;
01710     }
01711
01712     void translateException();
01713
01714     bool log();
01715     void react() const;
01716 };

```

```

01722
01723     namespace assertAction {
01724         enum Enum
01725         {
01726             nothing      = 0,
01727             dbgbreak     = 1,
01728             shouldthrow  = 2
01729         };
01730     } // namespace assertAction
01731
01732     DOCTEST_INTERFACE void failed_out_of_a_testing_context(const AssertData& ad);
01733
01734     DOCTEST_INTERFACE bool decomp_assert(assertType::Enum at, const char* file, int line,
01735                                         const char* expr, const Result& result);
01736
01737 #define DOCTEST_ASSERT_OUT_OF_TESTS(decomp)
01738     do {
01739         if(!is_running_in_test) {
01740             if(failed) {
01741                 ResultBuilder rb(at, file, line, expr);
01742                 rb.m_failed = failed;
01743                 rb.m_decomp = decomp;
01744                 failed_out_of_a_testing_context(rb);
01745                 if(isDebuggerActive() && !getContextOptions()->no_breaks)
01746                     DOCTEST_BREAK_INTO_DEBUGGER();
01747                 if(checkIfShouldThrow(at))
01748                     throwException();
01749             }
01750             return !failed;
01751         }
01752     } while(false)
01753
01754 #define DOCTEST_ASSERT_IN_TESTS(decomp)
01755     ResultBuilder rb(at, file, line, expr);
01756     rb.m_failed = failed;
01757     if(rb.m_failed || getContextOptions()->success)
01758         rb.m_decomp = decomp;
01759     if(rb.log())
01760         DOCTEST_BREAK_INTO_DEBUGGER();
01761     if(rb.m_failed && checkIfShouldThrow(at))
01762         throwException();
01763
01764     template <int comparison, typename L, typename R>
01765     DOCTEST_NOINLINE bool binary_assert(assertType::Enum at, const char* file, int line,
01766                                       const char* expr, const DOCTEST_REF_WRAP(L) lhs,
01767                                       const DOCTEST_REF_WRAP(R) rhs) {
01768         bool failed = !RelationalComparator<comparison, L, R>()(lhs, rhs);
01769
01770         // #####
01771         // IF THE DEBUGGER BREAKS HERE - GO 1 LEVEL UP IN THE CALLSTACK FOR THE FAILING ASSERT
01772         // THIS IS THE EFFECT OF HAVING 'DOCTEST_CONFIG_SUPER_FAST_ASSERTS' DEFINED
01773         // #####
01774         DOCTEST_ASSERT_OUT_OF_TESTS(stringifyBinaryExpr(lhs, " ", rhs));
01775         DOCTEST_ASSERT_IN_TESTS(stringifyBinaryExpr(lhs, " ", rhs));
01776         return !failed;
01777     }
01778
01779     template <typename L>
01780     DOCTEST_NOINLINE bool unary_assert(assertType::Enum at, const char* file, int line,
01781                                       const char* expr, const DOCTEST_REF_WRAP(L) val) {
01782         bool failed = !val;
01783
01784         if(at & assertType::is_false)
01785             failed = !failed;
01786
01787         // #####
01788         // IF THE DEBUGGER BREAKS HERE - GO 1 LEVEL UP IN THE CALLSTACK FOR THE FAILING ASSERT
01789         // THIS IS THE EFFECT OF HAVING 'DOCTEST_CONFIG_SUPER_FAST_ASSERTS' DEFINED
01790         // #####
01791         DOCTEST_ASSERT_OUT_OF_TESTS((DOCTEST_STRINGIFY(val)));
01792         DOCTEST_ASSERT_IN_TESTS((DOCTEST_STRINGIFY(val)));
01793         return !failed;
01794     }
01795
01796     struct DOCTEST_INTERFACE IExceptionTranslator
01797     {
01798         DOCTEST_DECLARE_INTERFACE(IExceptionTranslator)
01799         virtual bool translate(String& const = 0;
01800     };
01801
01802     template <typename T>
01803     class ExceptionTranslator : public IExceptionTranslator
01804     {
01805     public:
01806         explicit ExceptionTranslator(String (*translateFunction)(T))
01807             : m_translateFunction(translateFunction) {}
01808

```

```

01809         bool translate(String& res) const override {
01810 #ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
01811             try {
01812                 throw; // lgtm [cpp/rethrow-no-exception]
01813                 // cppcheck-suppress catchExceptionByValue
01814             } catch(const T& ex) {
01815                 res = m_translateFunction(ex);
01816                 return true;
01817             } catch(...) {}
01818 #endif
01819             static_cast<void>(res); // to silence -Wunused-parameter
01820             return false;
01821         }
01822
01823     private:
01824         String (*m_translateFunction)(T);
01825     };
01826
01827     DOCTEST_INTERFACE void registerExceptionTranslatorImpl(const IExceptionTranslator* et);
01828
01829     // ContextScope base class used to allow implementing methods of ContextScope
01830     // that don't depend on the template parameter in doctest.cpp.
01831     struct DOCTEST_INTERFACE ContextScopeBase : public IContextScope {
01832         ContextScopeBase(const ContextScopeBase&) = delete;
01833
01834         ContextScopeBase& operator=(const ContextScopeBase&) = delete;
01835         ContextScopeBase& operator=(ContextScopeBase&&) = delete;
01836
01837         ~ContextScopeBase() override = default;
01838
01839     protected:
01840         ContextScopeBase();
01841         ContextScopeBase(ContextScopeBase&& other) noexcept;
01842
01843         void destroy();
01844         bool need_to_destroy{true};
01845     };
01846
01847     template <typename L> class ContextScope : public ContextScopeBase
01848     {
01849     public:
01850         L lambda_;
01851
01852     public:
01853         explicit ContextScope(const L &lambda) : lambda_(lambda) {}
01854         explicit ContextScope(L&& lambda) : lambda_(static_cast<L&&>(lambda)) {}
01855
01856         ContextScope(const ContextScope&) = delete;
01857         ContextScope(ContextScope&&) noexcept = default;
01858
01859         ContextScope& operator=(const ContextScope&) = delete;
01860         ContextScope& operator=(ContextScope&&) = delete;
01861
01862         void stringify(std::ostream* s) const override { lambda_(s); }
01863
01864         ~ContextScope() override {
01865             if (need_to_destroy) {
01866                 destroy();
01867             }
01868         };
01869
01870     struct DOCTEST_INTERFACE MessageBuilder : public MessageData
01871     {
01872     public:
01873         std::ostream* m_stream;
01874         bool logged = false;
01875
01876         MessageBuilder(const char* file, int line, assertType::Enum severity);
01877
01878         MessageBuilder(const MessageBuilder&) = delete;
01879         MessageBuilder(MessageBuilder&&) = delete;
01880
01881         MessageBuilder& operator=(const MessageBuilder&) = delete;
01882         MessageBuilder& operator=(MessageBuilder&&) = delete;
01883
01884         ~MessageBuilder();
01885
01886         // the preferred way of chaining parameters for stringification
01887         DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4866)
01888         template <typename T>
01889         MessageBuilder& operator,(const T& in) {
01890             *m_stream << (DOCTEST_STRINGIFY(in));
01891             return *this;
01892         }
01893     };
01894     DOCTEST_MSVC_SUPPRESS_WARNING_POP
01895
01896     // kept here just for backwards-compatibility - the comma operator should be preferred now
01897     template <typename T>

```



```

01896     MessageBuilder& operator«(const T& in) { return this->operator,(in); }
01897
01898     // the `,' operator has the lowest operator precedence - if `«' is used by the user then
01899     // the `,' operator will be called last which is not what we want and thus the `*' operator
01900     // is used first (has higher operator precedence compared to `«') so that we guarantee that
01901     // an operator of the MessageBuilder class is called first before the rest of the parameters
01902     template <typename T>
01903     MessageBuilder& operator*(const T& in) { return this->operator,(in); }
01904
01905     bool log();
01906     void react();
01907 };
01908
01909     template <typename L>
01910     ContextScope<L> MakeContextScope(const L &lambda) {
01911         return ContextScope<L>(lambda);
01912     }
01913 } // namespace detail
01914
01915 #define DOCTEST_DEFINE_DECORATOR(name, type, def)
01916     struct name
01917     {
01918         type data;
01919         name(type in = def)
01920             : data(in) {}
01921         void fill(detail::TestCase& state) const { state.DOCTEST_CAT(m_, name) = data; }
01922         void fill(detail::TestSuite& state) const { state.DOCTEST_CAT(m_, name) = data; }
01923     }
01924
01925 DOCTEST_DEFINE_DECORATOR(test_suite, const char*, "");
01926 DOCTEST_DEFINE_DECORATOR(description, const char*, "");
01927 DOCTEST_DEFINE_DECORATOR(skip, bool, true);
01928 DOCTEST_DEFINE_DECORATOR(no_breaks, bool, true);
01929 DOCTEST_DEFINE_DECORATOR(no_output, bool, true);
01930 DOCTEST_DEFINE_DECORATOR(timeout, double, 0);
01931 DOCTEST_DEFINE_DECORATOR(may_fail, bool, true);
01932 DOCTEST_DEFINE_DECORATOR(should_fail, bool, true);
01933 DOCTEST_DEFINE_DECORATOR(expected_failures, int, 0);
01934
01935 template <typename T>
01936 int registerExceptionTranslator(String (*translateFunction)(T)) {
01937     DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wexit-time-destructors")
01938     static detail::ExceptionTranslator<T> exceptionTranslator(translateFunction);
01939     DOCTEST_CLANG_SUPPRESS_WARNING_POP
01940     detail::registerExceptionTranslatorImpl(&exceptionTranslator);
01941     return 0;
01942 }
01943
01944 } // namespace doctest
01945
01946 // in a separate namespace outside of doctest because the DOCTEST_TEST_SUITE macro
01947 // introduces an anonymous namespace in which getCurrentTestSuite gets overridden
01948 namespace doctest_detail_test_suite_ns {
01949     DOCTEST_INTERFACE doctest::detail::TestSuite& getCurrentTestSuite();
01950 } // namespace doctest_detail_test_suite_ns
01951
01952 namespace doctest {
01953     #else // DOCTEST_CONFIG_DISABLE
01954     template <typename T>
01955     int registerExceptionTranslator(String (*)(T)) {
01956         return 0;
01957     }
01958     #endif // DOCTEST_CONFIG_DISABLE
01959
01960     namespace detail {
01961         using assert_handler = void (*)(const AssertData&);
01962         struct ContextState;
01963     } // namespace detail
01964
01965     class DOCTEST_INTERFACE Context
01966     {
01967     public:
01968         detail::ContextState* p;
01969
01970         void parseArgs(int argc, const char* const* argv, bool withDefaults = false);
01971
01972         explicit Context(int argc = 0, const char* const* argv = nullptr);
01973
01974         Context(const Context&) = delete;
01975         Context(Context&&) = delete;
01976
01977         Context& operator=(const Context&) = delete;
01978         Context& operator=(Context&&) = delete;
01979
01980         ~Context(); // NOLINT(performance-trivially-destructible)
01981
01982         void applyCommandLine(int argc, const char* const* argv);

```

```

01983
01984     void addFilter(const char* filter, const char* value);
01985     void clearFilters();
01986     void setOption(const char* option, bool value);
01987     void setOption(const char* option, int value);
01988     void setOption(const char* option, const char* value);
01989
01990     bool shouldExit();
01991
01992     void setAsDefaultForAssertsOutOfTestCases();
01993
01994     void setAssertHandler(detail::assert_handler ah);
01995
01996     void setCout(std::ostream* out);
01997
01998     int run();
01999 };
02000
02001 namespace TestCaseFailureReason {
02002     enum Enum
02003     {
02004         None = 0,
02005         AssertFailure = 1, // an assertion has failed in the test case
02006         Exception = 2, // test case threw an exception
02007         Crash = 4, // a crash...
02008         TooManyFailedAsserts = 8, // the abort-after option
02009         Timeout = 16, // see the timeout decorator
02010         ShouldHaveFailedButDidnt = 32, // see the should_fail decorator
02011         ShouldHaveFailedAndDid = 64, // see the should_fail decorator
02012         DidntFailExactlyNumTimes = 128, // see the expected_failures decorator
02013         FailedExactlyNumTimes = 256, // see the expected_failures decorator
02014         CouldHaveFailedAndDid = 512 // see the may_fail decorator
02015     };
02016 } // namespace TestCaseFailureReason
02017
02018 struct DOCTEST_INTERFACE CurrentTestCaseStats
02019 {
02020     int numAssertsCurrentTest;
02021     int numAssertsFailedCurrentTest;
02022     double seconds;
02023     int failure_flags; // use TestCaseFailureReason::Enum
02024     bool testCaseSuccess;
02025 };
02026
02027 struct DOCTEST_INTERFACE TestCaseException
02028 {
02029     String error_string;
02030     bool is_crash;
02031 };
02032
02033 struct DOCTEST_INTERFACE TestRunStats
02034 {
02035     unsigned numTestCases;
02036     unsigned numTestCasesPassingFilters;
02037     unsigned numTestSuitesPassingFilters;
02038     unsigned numTestCasesFailed;
02039     int numAsserts;
02040     int numAssertsFailed;
02041 };
02042
02043 struct QueryData
02044 {
02045     const TestRunStats* run_stats = nullptr;
02046     const TestCaseData** data = nullptr;
02047     unsigned num_data = 0;
02048 };
02049
02050 struct DOCTEST_INTERFACE IReporter
02051 {
02052     // The constructor has to accept "const ContextOptions&" as a single argument
02053     // which has most of the options for the run + a pointer to the stdout stream
02054     // Reporter(const ContextOptions& in)
02055
02056     // called when a query should be reported (listing test cases, printing the version, etc.)
02057     virtual void report_query(const QueryData&) = 0;
02058
02059     // called when the whole test run starts
02060     virtual void test_run_start() = 0;
02061     // called when the whole test run ends (caching a pointer to the input doesn't make sense here)
02062     virtual void test_run_end(const TestRunStats&) = 0;
02063
02064     // called when a test case is started (safe to cache a pointer to the input)
02065     virtual void test_case_start(const TestCaseData&) = 0;
02066     // called when a test case is reentered because of unfinished subcases (safe to cache a pointer to
the input)
02067     virtual void test_case_reenter(const TestCaseData&) = 0;
02068     // called when a test case has ended

```

```

02069     virtual void test_case_end(const CurrentTestCaseStats&) = 0;
02070
02071     // called when an exception is thrown from the test case (or it crashes)
02072     virtual void test_case_exception(const TestCaseException&) = 0;
02073
02074     // called whenever a subcase is entered (don't cache pointers to the input)
02075     virtual void subcase_start(const SubcaseSignature&) = 0;
02076     // called whenever a subcase is exited (don't cache pointers to the input)
02077     virtual void subcase_end() = 0;
02078
02079     // called for each assert (don't cache pointers to the input)
02080     virtual void log_assert(const AssertData&) = 0;
02081     // called for each message (don't cache pointers to the input)
02082     virtual void log_message(const MessageData&) = 0;
02083
02084     // called when a test case is skipped either because it doesn't pass the filters, has a skip
decorator
02085     // or isn't in the execution range (between first and last) (safe to cache a pointer to the input)
02086     virtual void test_case_skipped(const TestCaseData&) = 0;
02087
02088     DOCTEST_DECLARE_INTERFACE(IReporter)
02089
02090     // can obtain all currently active contexts and stringify them if one wishes to do so
02091     static int get_num_active_contexts();
02092     static const IContextScope* const* get_active_contexts();
02093
02094     // can iterate through contexts which have been stringified automatically in their destructors
when an exception has been thrown
02095     static int get_num_stringified_contexts();
02096     static const String* get_stringified_contexts();
02097 };
02098
02099 namespace detail {
02100     using reporterCreatorFunc = IReporter* (*)(const ContextOptions&);
02101
02102     DOCTEST_INTERFACE void registerReporterImpl(const char* name, int prio, reporterCreatorFunc c,
bool isReporter);
02103
02104     template <typename Reporter>
02105     IReporter* reporterCreator(const ContextOptions& o) {
02106         return new Reporter(o);
02107     }
02108 } // namespace detail
02109
02110 template <typename Reporter>
02111 int registerReporter(const char* name, int priority, bool isReporter) {
02112     detail::registerReporterImpl(name, priority, detail::reporterCreator<Reporter>, isReporter);
02113     return 0;
02114 }
02115 } // namespace doctest
02116
02117 #ifdef DOCTEST_CONFIG_ASSERTS_RETURN_VALUES
02118 #define DOCTEST_FUNC_EMPTY [] { return false; }()
02119 #else
02120 #define DOCTEST_FUNC_EMPTY (void)0
02121 #endif
02122
02123 // if registering is not disabled
02124 #ifndef DOCTEST_CONFIG_DISABLE
02125
02126 #ifdef DOCTEST_CONFIG_ASSERTS_RETURN_VALUES
02127 #define DOCTEST_FUNC_SCOPE_BEGIN [&]
02128 #define DOCTEST_FUNC_SCOPE_END ()
02129 #define DOCTEST_FUNC_SCOPE_RET(v) return v
02130 #else
02131 #define DOCTEST_FUNC_SCOPE_BEGIN do
02132 #define DOCTEST_FUNC_SCOPE_END while(false)
02133 #define DOCTEST_FUNC_SCOPE_RET(v) (void)0
02134 #endif
02135
02136 // common code in asserts - for convenience
02137 #define DOCTEST_ASSERT_LOG_REACT_RETURN(b) \
02138     if(b.log()) DOCTEST_BREAK_INTO_DEBUGGER(); \
02139     b.react(); \
02140     DOCTEST_FUNC_SCOPE_RET(!b.m_failed)
02141
02142 #ifndef DOCTEST_CONFIG_NO_TRY_CATCH_IN_ASSERTS
02143 #define DOCTEST_WRAP_IN_TRY(x) x;
02144 #else // DOCTEST_CONFIG_NO_TRY_CATCH_IN_ASSERTS
02145 #define DOCTEST_WRAP_IN_TRY(x) \
02146     try { \
02147         x; \
02148     } catch(...) { DOCTEST_RB.translateException(); }
02149 #endif // DOCTEST_CONFIG_NO_TRY_CATCH_IN_ASSERTS
02150
02151 #ifndef DOCTEST_CONFIG_VOID_CAST_EXPRESSIONS
02152 #define DOCTEST_CAST_TO_VOID(...) \

```

```

02153     DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH("-Wuseless-cast")
02154     static_cast<void>(__VA_ARGS__);
02155     DOCTEST_GCC_SUPPRESS_WARNING_POP
02156 #else // DOCTEST_CONFIG_VOID_CAST_EXPRESSIONS
02157 #define DOCTEST_CAST_TO_VOID(...) __VA_ARGS__;
02158 #endif // DOCTEST_CONFIG_VOID_CAST_EXPRESSIONS
02159
02160 // registers the test by initializing a dummy var with a function
02161 #define DOCTEST_REGISTER_FUNCTION(global_prefix, f, decorators)
02162     global_prefix DOCTEST_GLOBAL_NO_WARNINGS(DOCTEST_ANONYMOUS(DOCTEST_ANON_VAR_), /* NOLINT */
02163         doctest::detail::regTest(
02164             doctest::detail::TestCase(
02165                 f, __FILE__, __LINE__,
02166                 doctest_detail_test_suite_ns::getCurrentTestSuite()) *
02167             decorators))
02168
02169 #define DOCTEST_IMPLEMENT_FIXTURE(der, base, func, decorators)
02170     namespace { /* NOLINT */
02171         struct der : public base
02172         {
02173             void f();
02174         };
02175         static DOCTEST_INLINE_NOINLINE void func() {
02176             der v;
02177             v.f();
02178         }
02179         DOCTEST_REGISTER_FUNCTION(DOCTEST_EMPTY, func, decorators)
02180     }
02181     DOCTEST_INLINE_NOINLINE void der::f() // NOLINT(misc-definitions-in-headers)
02182
02183 #define DOCTEST_CREATE_AND_REGISTER_FUNCTION(f, decorators)
02184     static void f();
02185     DOCTEST_REGISTER_FUNCTION(DOCTEST_EMPTY, f, decorators)
02186     static void f()
02187
02188 #define DOCTEST_CREATE_AND_REGISTER_FUNCTION_IN_CLASS(f, proxy, decorators)
02189     static doctest::detail::funcType proxy() { return f; }
02190     DOCTEST_REGISTER_FUNCTION(inline, proxy(), decorators)
02191     static void f()
02192
02193 // for registering tests
02194 #define DOCTEST_TEST_CASE(decorators)
02195     DOCTEST_CREATE_AND_REGISTER_FUNCTION(DOCTEST_ANONYMOUS(DOCTEST_ANON_FUNC_), decorators)
02196
02197 // for registering tests in classes - requires C++17 for inline variables!
02198 #if DOCTEST_CPLUSPLUS >= 201703L
02199 #define DOCTEST_TEST_CASE_CLASS(decorators)
02200     DOCTEST_CREATE_AND_REGISTER_FUNCTION_IN_CLASS(DOCTEST_ANONYMOUS(DOCTEST_ANON_FUNC_),
02201         DOCTEST_ANONYMOUS(DOCTEST_ANON_PROXY_),
02202         decorators)
02203 #else // DOCTEST_TEST_CASE_CLASS
02204 #define DOCTEST_TEST_CASE_CLASS(...)
02205     TEST_CASES_CAN_BE_REGISTERED_IN_CLASSES_ONLY_IN_CPP17_MODE_OR_WITH_VS_2017_OR_NEWER
02206 #endif // DOCTEST_TEST_CASE_CLASS
02207
02208 // for registering tests with a fixture
02209 #define DOCTEST_TEST_CASE_FIXTURE(c, decorators)
02210     DOCTEST_IMPLEMENT_FIXTURE(DOCTEST_ANONYMOUS(DOCTEST_ANON_CLASS_), c,
02211         DOCTEST_ANONYMOUS(DOCTEST_ANON_FUNC_), decorators)
02212
02213 // for converting types to strings without the <typeinfo> header and demangling
02214 #define DOCTEST_TYPE_TO_STRING_AS(str, ...)
02215     namespace doctest {
02216         template <>
02217         inline String toString<__VA_ARGS__>() {
02218             return str;
02219         }
02220     }
02221     static_assert(true, "")
02222
02223 #define DOCTEST_TYPE_TO_STRING(...) DOCTEST_TYPE_TO_STRING_AS(#__VA_ARGS__, __VA_ARGS__)
02224
02225 #define DOCTEST_TEST_CASE_TEMPLATE_DEFINE_IMPL(dec, T, iter, func)
02226     template <typename T>
02227     static void func();
02228     namespace { /* NOLINT */
02229         template <typename Tuple>
02230         struct iter;
02231         template <typename Type, typename... Rest>
02232         struct iter<std::tuple<Type, Rest...>
02233         {
02234             iter(const char* file, unsigned line, int index) {
02235                 doctest::detail::regTest(doctest::detail::TestCase(func<Type>, file, line,
02236                     doctest_detail_test_suite_ns::getCurrentTestSuite(),
02237                     doctest::toString<Type>(),
02238                     int(line) * 1000 + index)
02239                     * dec);

```

```

02240         iter<std::tuple<Rest...>>(file, line, index + 1);
02241     }
02242 };
02243 template <>
02244 struct iter<std::tuple<>>
02245 {
02246     iter(const char*, unsigned, int) {}
02247 };
02248 }
02249 template <typename T>
02250 static void func()
02251
02252 #define DOCTEST_TEST_CASE_TEMPLATE_DEFINE(dec, T, id)
02253     DOCTEST_TEST_CASE_TEMPLATE_DEFINE_IMPL(dec, T, DOCTEST_CAT(id, ITERATOR),
02254     DOCTEST_ANONYMOUS(DOCTEST_ANON_TMP_))
02255
02256 #define DOCTEST_TEST_CASE_TEMPLATE_INSTANTIATE_IMPL(id, anon, ...)
02257     DOCTEST_GLOBAL_NO_WARNINGS(DOCTEST_CAT(anon, DUMMY), /* NOLINT(cert-err58-cpp,
fuchsia-statically-constructed-objects) */ \
02258     doctest::detail::instantiationHelper(
02259     DOCTEST_CAT(id, ITERATOR)<__VA_ARGS__>(__FILE__, __LINE__, 0)))
02260
02261 #define DOCTEST_TEST_CASE_TEMPLATE_INVOKE(id, ...)
02262     DOCTEST_TEST_CASE_TEMPLATE_INSTANTIATE_IMPL(id, DOCTEST_ANONYMOUS(DOCTEST_ANON_TMP_),
std::tuple<__VA_ARGS__>) \
02263     static_assert(true, "")
02264
02265 #define DOCTEST_TEST_CASE_TEMPLATE_APPLY(id, ...)
02266     DOCTEST_TEST_CASE_TEMPLATE_INSTANTIATE_IMPL(id, DOCTEST_ANONYMOUS(DOCTEST_ANON_TMP_), __VA_ARGS__)
\
02267     static_assert(true, "")
02268
02269 #define DOCTEST_TEST_CASE_TEMPLATE_IMPL(dec, T, anon, ...)
02270     DOCTEST_TEST_CASE_TEMPLATE_DEFINE_IMPL(dec, T, DOCTEST_CAT(anon, ITERATOR), anon);
02271     DOCTEST_TEST_CASE_TEMPLATE_INSTANTIATE_IMPL(anon, anon, std::tuple<__VA_ARGS__>)
02272     template <typename T>
02273     static void anon()
02274
02275 #define DOCTEST_TEST_CASE_TEMPLATE(dec, T, ...)
02276     DOCTEST_TEST_CASE_TEMPLATE_IMPL(dec, T, DOCTEST_ANONYMOUS(DOCTEST_ANON_TMP_), __VA_ARGS__)
02277
02278 // for subcases
02279 #define DOCTEST_SUBCASE(name)
02280     if(const doctest::detail::Subcase & DOCTEST_ANONYMOUS(DOCTEST_ANON_SUBCASE_) DOCTEST_UNUSED =
02281     doctest::detail::Subcase(name, __FILE__, __LINE__))
02282
02283 // for grouping tests in test suites by using code blocks
02284 #define DOCTEST_TEST_SUITE_IMPL(decorators, ns_name)
02285     namespace ns_name { namespace doctest_detail_test_suite_ns {
02286         static DOCTEST_NOINLINE doctest::detail::TestSuite& getCurrentTestSuite() noexcept {
02287             DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4640)
02288             DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wexit-time-destructors")
02289             DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH("-Wmissing-field-initializers")
02290             static doctest::detail::TestSuite data{};
02291             static bool        initd = false;
02292             DOCTEST_MSVC_SUPPRESS_WARNING_POP
02293             DOCTEST_CLANG_SUPPRESS_WARNING_POP
02294             DOCTEST_GCC_SUPPRESS_WARNING_POP
02295             if(!initd) {
02296                 data* decorators;
02297                 initd = true;
02298             }
02299             return data;
02300         }
02301     }
02302 }
02303 namespace ns_name
02304
02305 #define DOCTEST_TEST_SUITE(decorators)
02306     DOCTEST_TEST_SUITE_IMPL(decorators, DOCTEST_ANONYMOUS(DOCTEST_ANON_SUITE_))
02307
02308 // for starting a testsuite block
02309 #define DOCTEST_TEST_SUITE_BEGIN(decorators)
02310     DOCTEST_GLOBAL_NO_WARNINGS(DOCTEST_ANONYMOUS(DOCTEST_ANON_VAR_), /* NOLINT(cert-err58-cpp) */
doctest::detail::setTestSuite(doctest::detail::TestSuite() * decorators))
02311     static_assert(true, "")
02312
02313 // for ending a testsuite block
02314 #define DOCTEST_TEST_SUITE_END
02315     DOCTEST_GLOBAL_NO_WARNINGS(DOCTEST_ANONYMOUS(DOCTEST_ANON_VAR_), /* NOLINT(cert-err58-cpp) */
doctest::detail::setTestSuite(doctest::detail::TestSuite() * ""))
02316     using DOCTEST_ANONYMOUS(DOCTEST_ANON_FOR_SEMICOLON_) = int
02317
02318 // for registering exception translators
02319 #define DOCTEST_REGISTER_EXCEPTION_TRANSLATOR_IMPL(translatorName, signature)
02320     inline doctest::String translatorName(signature);
02321     DOCTEST_GLOBAL_NO_WARNINGS(DOCTEST_ANONYMOUS(DOCTEST_ANON_TRANSLATOR_), /* NOLINT(cert-err58-cpp)

```

```

2324 */ \
2325     doctest::registerExceptionTranslator(translatorName) \
2326     doctest::String translatorName(signature) \
2327 #define DOCTEST_REGISTER_EXCEPTION_TRANSLATOR(signature) \
2328     DOCTEST_REGISTER_EXCEPTION_TRANSLATOR_IMPL(DOCTEST_ANONYMOUS(DOCTEST_ANON_TRANSLATOR_), \
2329     signature) \
2330
2331 // for registering reporters
2332 #define DOCTEST_REGISTER_REPORTER(name, priority, reporter) \
2333     DOCTEST_GLOBAL_NO_WARNINGS(DOCTEST_ANONYMOUS(DOCTEST_ANON_REPORTER_), /* NOLINT(cert-err58-cpp) */ \
2334 \
2335     doctest::registerReporter<reporter>(name, priority, true)) \
2336     static_assert(true, "") \
2337 // for registering listeners
2338 #define DOCTEST_REGISTER_LISTENER(name, priority, reporter) \
2339     DOCTEST_GLOBAL_NO_WARNINGS(DOCTEST_ANONYMOUS(DOCTEST_ANON_REPORTER_), /* NOLINT(cert-err58-cpp) */ \
2340 \
2341     doctest::registerReporter<reporter>(name, priority, false)) \
2342     static_assert(true, "") \
2343 // clang-format off
2344 // for logging - disabling formatting because it's important to have these on 2 separate lines - see
2345 // PR #557
2346 #define DOCTEST_INFO(...) \
2347     DOCTEST_INFO_IMPL(DOCTEST_ANONYMOUS(DOCTEST_CAPTURE_), \
2348     DOCTEST_ANONYMOUS(DOCTEST_CAPTURE_OTHER_), \
2349     __VA_ARGS__) \
2350 // clang-format on
2351 #define DOCTEST_INFO_IMPL(mb_name, s_name, ...) \
2352     auto DOCTEST_ANONYMOUS(DOCTEST_CAPTURE_) = doctest::detail::MakeContextScope( \
2353     [&](std::ostream* s_name) { \
2354         doctest::detail::MessageBuilder mb_name(__FILE__, __LINE__, doctest::assertType::is_warn); \
2355         mb_name.m_stream = s_name; \
2356         mb_name * __VA_ARGS__; \
2357     }) \
2358
2359 #define DOCTEST_CAPTURE(x) DOCTEST_INFO(#x " := ", x)
2360
2361 #define DOCTEST_ADD_AT_IMPL(type, file, line, mb, ...) \
2362     DOCTEST_FUNC_SCOPE_BEGIN { \
2363         doctest::detail::MessageBuilder mb(file, line, doctest::assertType::type); \
2364         mb * __VA_ARGS__; \
2365         if(mb.log()) \
2366             DOCTEST_BREAK_INTO_DEBUGGER(); \
2367         mb.react(); \
2368     } DOCTEST_FUNC_SCOPE_END \
2369 // clang-format off
2370 #define DOCTEST_ADD_MESSAGE_AT(file, line, ...) DOCTEST_ADD_AT_IMPL(is_warn, file, line, \
2371 DOCTEST_ANONYMOUS(DOCTEST_MESSAGE_), __VA_ARGS__) \
2372 #define DOCTEST_ADD_FAIL_CHECK_AT(file, line, ...) DOCTEST_ADD_AT_IMPL(is_check, file, line, \
2373 DOCTEST_ANONYMOUS(DOCTEST_MESSAGE_), __VA_ARGS__) \
2374 #define DOCTEST_ADD_FAIL_AT(file, line, ...) DOCTEST_ADD_AT_IMPL(is_require, file, line, \
2375 DOCTEST_ANONYMOUS(DOCTEST_MESSAGE_), __VA_ARGS__) \
2376 // clang-format on
2377 #define DOCTEST_MESSAGE(...) DOCTEST_ADD_MESSAGE_AT(__FILE__, __LINE__, __VA_ARGS__) \
2378 #define DOCTEST_FAIL_CHECK(...) DOCTEST_ADD_FAIL_CHECK_AT(__FILE__, __LINE__, __VA_ARGS__) \
2379 #define DOCTEST_FAIL(...) DOCTEST_ADD_FAIL_AT(__FILE__, __LINE__, __VA_ARGS__) \
2380 #define DOCTEST_TO_LVALUE(...) __VA_ARGS__ // Not removed to keep backwards compatibility.
2381
2382 #ifndef DOCTEST_CONFIG_SUPER_FAST_ASSERTS
2383
2384 #define DOCTEST_ASSERT_IMPLEMENT_2(assert_type, ...) \
2385     DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Woverloaded-shift-op-parentheses") \
2386     /* NOLINTNEXTLINE(clang-analyzer-cplusplus.NewDeleteLeaks) */ \
2387     doctest::detail::ResultBuilder DOCTEST_RB(doctest::assertType::assert_type, __FILE__, \
2388     __LINE__, #__VA_ARGS__); \
2389     DOCTEST_WRAP_IN_TRY(DOCTEST_RB.setResult( \
2390     doctest::detail::ExpressionDecomposer(doctest::assertType::assert_type) \
2391     « __VA_ARGS__)) /* NOLINTNEXTLINE(clang-analyzer-cplusplus.NewDeleteLeaks) */ \
2392     DOCTEST_ASSERT_LOG_REACT_RETURN(DOCTEST_RB) \
2393     DOCTEST_CLANG_SUPPRESS_WARNING_POP \
2394
2395 #define DOCTEST_ASSERT_IMPLEMENT_1(assert_type, ...) \
2396     DOCTEST_FUNC_SCOPE_BEGIN { \
2397         DOCTEST_ASSERT_IMPLEMENT_2(assert_type, __VA_ARGS__); \
2398     } DOCTEST_FUNC_SCOPE_END // NOLINT(clang-analyzer-cplusplus.NewDeleteLeaks) \
2399
2400 #define DOCTEST_BINARY_ASSERT(assert_type, comp, ...) \
2401     DOCTEST_FUNC_SCOPE_BEGIN { \
2402         doctest::detail::ResultBuilder DOCTEST_RB(doctest::assertType::assert_type, __FILE__, \
2403         __LINE__, #__VA_ARGS__); \

```

```

02404         DOCTEST_WRAP_IN_TRY(
02405             DOCTEST_RB.binary_assert<doctest::detail::binaryAssertComparison::comp>(
02406                 __VA_ARGS__)
02407         DOCTEST_ASSERT_LOG_REACT_RETURN(DOCTEST_RB);
02408     } DOCTEST_FUNC_SCOPE_END
02409
02410 #define DOCTEST_UNARY_ASSERT(assert_type, ...)
02411     DOCTEST_FUNC_SCOPE_BEGIN {
02412         doctest::detail::ResultBuilder DOCTEST_RB(doctest::assertType::assert_type, __FILE__,
02413             __LINE__, #__VA_ARGS__);
02414         DOCTEST_WRAP_IN_TRY(DOCTEST_RB.unary_assert(__VA_ARGS__))
02415         DOCTEST_ASSERT_LOG_REACT_RETURN(DOCTEST_RB);
02416     } DOCTEST_FUNC_SCOPE_END
02417
02418 #else // DOCTEST_CONFIG_SUPER_FAST_ASSERTS
02419
02420 // necessary for <ASSERT>_MESSAGE
02421 #define DOCTEST_ASSERT_IMPLEMENT_2 DOCTEST_ASSERT_IMPLEMENT_1
02422
02423 #define DOCTEST_ASSERT_IMPLEMENT_1(assert_type, ...)
02424     DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Woverloaded-shift-op-parentheses")
02425     doctest::detail::decomp_assert(
02426         doctest::assertType::assert_type, __FILE__, __LINE__, #__VA_ARGS__,
02427         doctest::detail::ExpressionComposer(doctest::assertType::assert_type)
02428         « __VA_ARGS__ DOCTEST_CLANG_SUPPRESS_WARNING_POP
02429
02430 #define DOCTEST_BINARY_ASSERT(assert_type, comparison, ...)
02431     doctest::detail::binary_assert<doctest::detail::binaryAssertComparison::comparison>(
02432         doctest::assertType::assert_type, __FILE__, __LINE__, #__VA_ARGS__, __VA_ARGS__)
02433
02434 #define DOCTEST_UNARY_ASSERT(assert_type, ...)
02435     doctest::detail::unary_assert(doctest::assertType::assert_type, __FILE__, __LINE__,
02436         #__VA_ARGS__, __VA_ARGS__)
02437
02438 #endif // DOCTEST_CONFIG_SUPER_FAST_ASSERTS
02439
02440 #define DOCTEST_WARN(...) DOCTEST_ASSERT_IMPLEMENT_1(DT_WARN, __VA_ARGS__)
02441 #define DOCTEST_CHECK(...) DOCTEST_ASSERT_IMPLEMENT_1(DT_CHECK, __VA_ARGS__)
02442 #define DOCTEST_REQUIRE(...) DOCTEST_ASSERT_IMPLEMENT_1(DT_REQUIRE, __VA_ARGS__)
02443 #define DOCTEST_WARN_FALSE(...) DOCTEST_ASSERT_IMPLEMENT_1(DT_WARN_FALSE, __VA_ARGS__)
02444 #define DOCTEST_CHECK_FALSE(...) DOCTEST_ASSERT_IMPLEMENT_1(DT_CHECK_FALSE, __VA_ARGS__)
02445 #define DOCTEST_REQUIRE_FALSE(...) DOCTEST_ASSERT_IMPLEMENT_1(DT_REQUIRE_FALSE, __VA_ARGS__)
02446
02447 // clang-format off
02448 #define DOCTEST_WARN_MESSAGE(cond, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
02449     DOCTEST_ASSERT_IMPLEMENT_2(DT_WARN, cond); } DOCTEST_FUNC_SCOPE_END
02450 #define DOCTEST_CHECK_MESSAGE(cond, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
02451     DOCTEST_ASSERT_IMPLEMENT_2(DT_CHECK, cond); } DOCTEST_FUNC_SCOPE_END
02452 #define DOCTEST_REQUIRE_MESSAGE(cond, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
02453     DOCTEST_ASSERT_IMPLEMENT_2(DT_REQUIRE, cond); } DOCTEST_FUNC_SCOPE_END
02454 #define DOCTEST_WARN_FALSE_MESSAGE(cond, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
02455     DOCTEST_ASSERT_IMPLEMENT_2(DT_WARN_FALSE, cond); } DOCTEST_FUNC_SCOPE_END
02456 #define DOCTEST_CHECK_FALSE_MESSAGE(cond, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
02457     DOCTEST_ASSERT_IMPLEMENT_2(DT_CHECK_FALSE, cond); } DOCTEST_FUNC_SCOPE_END
02458 #define DOCTEST_REQUIRE_FALSE_MESSAGE(cond, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
02459     DOCTEST_ASSERT_IMPLEMENT_2(DT_REQUIRE_FALSE, cond); } DOCTEST_FUNC_SCOPE_END
02460 // clang-format on
02461
02462 #define DOCTEST_WARN_EQ(...) DOCTEST_BINARY_ASSERT(DT_WARN_EQ, eq, __VA_ARGS__)
02463 #define DOCTEST_CHECK_EQ(...) DOCTEST_BINARY_ASSERT(DT_CHECK_EQ, eq, __VA_ARGS__)
02464 #define DOCTEST_REQUIRE_EQ(...) DOCTEST_BINARY_ASSERT(DT_REQUIRE_EQ, eq, __VA_ARGS__)
02465 #define DOCTEST_WARN_NE(...) DOCTEST_BINARY_ASSERT(DT_WARN_NE, ne, __VA_ARGS__)
02466 #define DOCTEST_CHECK_NE(...) DOCTEST_BINARY_ASSERT(DT_CHECK_NE, ne, __VA_ARGS__)
02467 #define DOCTEST_REQUIRE_NE(...) DOCTEST_BINARY_ASSERT(DT_REQUIRE_NE, ne, __VA_ARGS__)
02468 #define DOCTEST_WARN_GT(...) DOCTEST_BINARY_ASSERT(DT_WARN_GT, gt, __VA_ARGS__)
02469 #define DOCTEST_CHECK_GT(...) DOCTEST_BINARY_ASSERT(DT_CHECK_GT, gt, __VA_ARGS__)
02470 #define DOCTEST_REQUIRE_GT(...) DOCTEST_BINARY_ASSERT(DT_REQUIRE_GT, gt, __VA_ARGS__)
02471 #define DOCTEST_WARN_LT(...) DOCTEST_BINARY_ASSERT(DT_WARN_LT, lt, __VA_ARGS__)
02472 #define DOCTEST_CHECK_LT(...) DOCTEST_BINARY_ASSERT(DT_CHECK_LT, lt, __VA_ARGS__)
02473 #define DOCTEST_REQUIRE_LT(...) DOCTEST_BINARY_ASSERT(DT_REQUIRE_LT, lt, __VA_ARGS__)
02474 #define DOCTEST_WARN_GE(...) DOCTEST_BINARY_ASSERT(DT_WARN_GE, ge, __VA_ARGS__)
02475 #define DOCTEST_CHECK_GE(...) DOCTEST_BINARY_ASSERT(DT_CHECK_GE, ge, __VA_ARGS__)
02476 #define DOCTEST_REQUIRE_GE(...) DOCTEST_BINARY_ASSERT(DT_REQUIRE_GE, ge, __VA_ARGS__)
02477 #define DOCTEST_WARN_LE(...) DOCTEST_BINARY_ASSERT(DT_WARN_LE, le, __VA_ARGS__)
02478 #define DOCTEST_CHECK_LE(...) DOCTEST_BINARY_ASSERT(DT_CHECK_LE, le, __VA_ARGS__)
02479 #define DOCTEST_REQUIRE_LE(...) DOCTEST_BINARY_ASSERT(DT_REQUIRE_LE, le, __VA_ARGS__)
02480
02481 #define DOCTEST_WARN_UNARY(...) DOCTEST_UNARY_ASSERT(DT_WARN_UNARY, __VA_ARGS__)
02482 #define DOCTEST_CHECK_UNARY(...) DOCTEST_UNARY_ASSERT(DT_CHECK_UNARY, __VA_ARGS__)
02483 #define DOCTEST_REQUIRE_UNARY(...) DOCTEST_UNARY_ASSERT(DT_REQUIRE_UNARY, __VA_ARGS__)
02484 #define DOCTEST_WARN_UNARY_FALSE(...) DOCTEST_UNARY_ASSERT(DT_WARN_UNARY_FALSE, __VA_ARGS__)
02485 #define DOCTEST_CHECK_UNARY_FALSE(...) DOCTEST_UNARY_ASSERT(DT_CHECK_UNARY_FALSE, __VA_ARGS__)
02486 #define DOCTEST_REQUIRE_UNARY_FALSE(...) DOCTEST_UNARY_ASSERT(DT_REQUIRE_UNARY_FALSE, __VA_ARGS__)
02487
02488 #ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
02489
02490 #define DOCTEST_ASSERT_THROWS_AS(expr, assert_type, message, ...)

```



```

02485     DOCTEST_FUNC_SCOPE_BEGIN {
02486         if(!doctest::getContextOptions()->no_throw) {
02487             doctest::detail::ResultBuilder DOCTEST_RB(doctest::assertType::assert_type, __FILE__,
02488                 __LINE__, #expr, #__VA_ARGS__, message);
02489             try {
02490                 DOCTEST_CAST_TO_VOID(expr)
02491             } catch(const typename doctest::detail::types::remove_const<
02492                 typename doctest::detail::types::remove_reference<__VA_ARGS__>::type>::type&) {
02493                 DOCTEST_RB.translateException();
02494                 DOCTEST_RB.m_threw_as = true;
02495             } catch(...) { DOCTEST_RB.translateException(); }
02496             DOCTEST_ASSERT_LOG_REACT_RETURN(DOCTEST_RB);
02497         } else { /* NOLINT(*-else-after-return) */
02498             DOCTEST_FUNC_SCOPE_RET(false);
02499         }
02500     } DOCTEST_FUNC_SCOPE_END
02501
02502 #define DOCTEST_ASSERT_THROWS_WITH(expr, expr_str, assert_type, ...)
02503     DOCTEST_FUNC_SCOPE_BEGIN {
02504         if(!doctest::getContextOptions()->no_throw) {
02505             doctest::detail::ResultBuilder DOCTEST_RB(doctest::assertType::assert_type, __FILE__,
02506                 __LINE__, expr_str, "", __VA_ARGS__);
02507             try {
02508                 DOCTEST_CAST_TO_VOID(expr)
02509             } catch(...) { DOCTEST_RB.translateException(); }
02510             DOCTEST_ASSERT_LOG_REACT_RETURN(DOCTEST_RB);
02511         } else { /* NOLINT(*-else-after-return) */
02512             DOCTEST_FUNC_SCOPE_RET(false);
02513         }
02514     } DOCTEST_FUNC_SCOPE_END
02515
02516 #define DOCTEST_ASSERT_NOTHROW(assert_type, ...)
02517     DOCTEST_FUNC_SCOPE_BEGIN {
02518         doctest::detail::ResultBuilder DOCTEST_RB(doctest::assertType::assert_type, __FILE__,
02519             __LINE__, #__VA_ARGS__);
02520         try {
02521             DOCTEST_CAST_TO_VOID(__VA_ARGS__)
02522         } catch(...) { DOCTEST_RB.translateException(); }
02523         DOCTEST_ASSERT_LOG_REACT_RETURN(DOCTEST_RB);
02524     } DOCTEST_FUNC_SCOPE_END
02525
02526 // clang-format off
02527 #define DOCTEST_WARN_THROWS(...) DOCTEST_ASSERT_THROWS_WITH((__VA_ARGS__), #__VA_ARGS__,
02528     DT_WARN_THROWS, "")
02529 #define DOCTEST_CHECK_THROWS(...) DOCTEST_ASSERT_THROWS_WITH((__VA_ARGS__), #__VA_ARGS__,
02530     DT_CHECK_THROWS, "")
02531 #define DOCTEST_REQUIRE_THROWS(...) DOCTEST_ASSERT_THROWS_WITH((__VA_ARGS__), #__VA_ARGS__,
02532     DT_REQUIRE_THROWS, "")
02533
02534 #define DOCTEST_WARN_THROWS_AS(expr, ...) DOCTEST_ASSERT_THROWS_AS(expr, DT_WARN_THROWS_AS, "",
02535     __VA_ARGS__)
02536 #define DOCTEST_CHECK_THROWS_AS(expr, ...) DOCTEST_ASSERT_THROWS_AS(expr, DT_CHECK_THROWS_AS, "",
02537     __VA_ARGS__)
02538 #define DOCTEST_REQUIRE_THROWS_AS(expr, ...) DOCTEST_ASSERT_THROWS_AS(expr, DT_REQUIRE_THROWS_AS, "",
02539     __VA_ARGS__)
02540
02541 #define DOCTEST_WARN_THROWS_WITH(expr, ...) DOCTEST_ASSERT_THROWS_WITH(expr, #expr,
02542     DT_WARN_THROWS_WITH, __VA_ARGS__)
02543 #define DOCTEST_CHECK_THROWS_WITH(expr, ...) DOCTEST_ASSERT_THROWS_WITH(expr, #expr,
02544     DT_CHECK_THROWS_WITH, __VA_ARGS__)
02545 #define DOCTEST_REQUIRE_THROWS_WITH(expr, ...) DOCTEST_ASSERT_THROWS_WITH(expr, #expr,
02546     DT_REQUIRE_THROWS_WITH, __VA_ARGS__)
02547
02548 #define DOCTEST_WARN_THROWS_WITH_AS(expr, message, ...) DOCTEST_ASSERT_THROWS_AS(expr,
02549     DT_WARN_THROWS_WITH_AS, message, __VA_ARGS__)
02550 #define DOCTEST_CHECK_THROWS_WITH_AS(expr, message, ...) DOCTEST_ASSERT_THROWS_AS(expr,
02551     DT_CHECK_THROWS_WITH_AS, message, __VA_ARGS__)
02552 #define DOCTEST_REQUIRE_THROWS_WITH_AS(expr, message, ...) DOCTEST_ASSERT_THROWS_AS(expr,
02553     DT_REQUIRE_THROWS_WITH_AS, message, __VA_ARGS__)
02554
02555 #define DOCTEST_WARN_NOTHROW(...) DOCTEST_ASSERT_NOTHROW(DT_WARN_NOTHROW, __VA_ARGS__)
02556 #define DOCTEST_CHECK_NOTHROW(...) DOCTEST_ASSERT_NOTHROW(DT_CHECK_NOTHROW, __VA_ARGS__)
02557 #define DOCTEST_REQUIRE_NOTHROW(...) DOCTEST_ASSERT_NOTHROW(DT_REQUIRE_NOTHROW, __VA_ARGS__)
02558
02559 #define DOCTEST_WARN_THROWS_MESSAGE(expr, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
02560     DOCTEST_WARN_THROWS(expr); } DOCTEST_FUNC_SCOPE_END
02561 #define DOCTEST_CHECK_THROWS_MESSAGE(expr, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
02562     DOCTEST_CHECK_THROWS(expr); } DOCTEST_FUNC_SCOPE_END
02563 #define DOCTEST_REQUIRE_THROWS_MESSAGE(expr, ...) DOCTEST_FUNC_SCOPE_BEGIN {
02564     DOCTEST_INFO(__VA_ARGS__); DOCTEST_REQUIRE_THROWS(expr); } DOCTEST_FUNC_SCOPE_END
02565 #define DOCTEST_WARN_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_FUNC_SCOPE_BEGIN {
02566     DOCTEST_INFO(__VA_ARGS__); DOCTEST_WARN_THROWS_AS(expr, ex); } DOCTEST_FUNC_SCOPE_END
02567 #define DOCTEST_CHECK_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_FUNC_SCOPE_BEGIN {
02568     DOCTEST_INFO(__VA_ARGS__); DOCTEST_CHECK_THROWS_AS(expr, ex); } DOCTEST_FUNC_SCOPE_END
02569 #define DOCTEST_REQUIRE_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_FUNC_SCOPE_BEGIN {
02570     DOCTEST_INFO(__VA_ARGS__); DOCTEST_REQUIRE_THROWS_AS(expr, ex); } DOCTEST_FUNC_SCOPE_END
02571 #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
02554 #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
02555 #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
02556 #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
02557 #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
02558 #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
02559 #define DOCTEST_WARN_NOTHROW_MESSAGE(expr, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
DOCTEST_WARN_NOTHROW(expr); } DOCTEST_FUNC_SCOPE_END
02560 #define DOCTEST_CHECK_NOTHROW_MESSAGE(expr, ...) DOCTEST_FUNC_SCOPE_BEGIN { DOCTEST_INFO(__VA_ARGS__);
DOCTEST_CHECK_NOTHROW(expr); } DOCTEST_FUNC_SCOPE_END
02561 #define DOCTEST_REQUIRE_NOTHROW_MESSAGE(expr, ...) DOCTEST_FUNC_SCOPE_BEGIN {
DOCTEST_INFO(__VA_ARGS__); DOCTEST_REQUIRE_NOTHROW(expr); } DOCTEST_FUNC_SCOPE_END
02562 // clang-format on
02563
02564 #endif // DOCTEST_CONFIG_NO_EXCEPTIONS
02565
02566 // =====
02567 // == WHAT FOLLOWS IS VERSIONS OF THE MACROS THAT DO NOT DO ANY REGISTERING! ==
02568 // == THIS CAN BE ENABLED BY DEFINING DOCTEST_CONFIG_DISABLE GLOBALLY! ==
02569 // =====
02570 #else // DOCTEST_CONFIG_DISABLE
02571
02572 #define DOCTEST_IMPLEMENT_FIXTURE(der, base, func, name) \
02573     namespace /* NOLINT */ { \
02574         template <typename DOCTEST_UNUSED_TEMPLATE_TYPE> \
02575             struct der : public base \
02576             { void f(); }; \
02577     } \
02578     template <typename DOCTEST_UNUSED_TEMPLATE_TYPE> \
02579     inline void der<DOCTEST_UNUSED_TEMPLATE_TYPE>::f() \
02580
02581 #define DOCTEST_CREATE_AND_REGISTER_FUNCTION(f, name) \
02582     template <typename DOCTEST_UNUSED_TEMPLATE_TYPE> \
02583     static inline void f() \
02584
02585 // for registering tests
02586 #define DOCTEST_TEST_CASE(name) \
02587     DOCTEST_CREATE_AND_REGISTER_FUNCTION(DOCTEST_ANONYMOUS(DOCTEST_ANON_FUNC_), name) \
02588
02589 // for registering tests in classes
02590 #define DOCTEST_TEST_CASE_CLASS(name) \
02591     DOCTEST_CREATE_AND_REGISTER_FUNCTION(DOCTEST_ANONYMOUS(DOCTEST_ANON_FUNC_), name) \
02592
02593 // for registering tests with a fixture
02594 #define DOCTEST_TEST_CASE_FIXTURE(x, name) \
02595     DOCTEST_IMPLEMENT_FIXTURE(DOCTEST_ANONYMOUS(DOCTEST_ANON_CLASS_), x, \
02596         DOCTEST_ANONYMOUS(DOCTEST_ANON_FUNC_), name) \
02597
02598 // for converting types to strings without the <typeinfo> header and demangling
02599 #define DOCTEST_TYPE_TO_STRING_AS(str, ...) static_assert(true, "")
02600 #define DOCTEST_TYPE_TO_STRING(...) static_assert(true, "")
02601
02602 // for typed tests
02603 #define DOCTEST_TEST_CASE_TEMPLATE(name, type, ...) \
02604     template <typename type> \
02605     inline void DOCTEST_ANONYMOUS(DOCTEST_ANON_TMP_()) \
02606
02607 #define DOCTEST_TEST_CASE_TEMPLATE_DEFINE(name, type, id) \
02608     template <typename type> \
02609     inline void DOCTEST_ANONYMOUS(DOCTEST_ANON_TMP_()) \
02610
02611 #define DOCTEST_TEST_CASE_TEMPLATE_INVOKE(id, ...) static_assert(true, "")
02612 #define DOCTEST_TEST_CASE_TEMPLATE_APPLY(id, ...) static_assert(true, "")
02613
02614 // for subcases
02615 #define DOCTEST_SUBCASE(name)
02616
02617 // for a testsuite block
02618 #define DOCTEST_TEST_SUITE(name) namespace // NOLINT
02619
02620 // for starting a testsuite block
02621 #define DOCTEST_TEST_SUITE_BEGIN(name) static_assert(true, "")
02622
02623 // for ending a testsuite block
02624 #define DOCTEST_TEST_SUITE_END using DOCTEST_ANONYMOUS(DOCTEST_ANON_FOR_SEMICOLON_) = int
02625
02626 #define DOCTEST_REGISTER_EXCEPTION_TRANSLATOR(signature) \
02627     template <typename DOCTEST_UNUSED_TEMPLATE_TYPE> \
02628     static inline doctest::String DOCTEST_ANONYMOUS(DOCTEST_ANON_TRANSLATOR_)(signature) \
02629
02630 #define DOCTEST_REGISTER_REPORTER(name, priority, reporter)
02631 #define DOCTEST_REGISTER_LISTENER(name, priority, reporter)

```

```

02632
02633 #define DOCTEST_INFO(...) (static_cast<void>(0))
02634 #define DOCTEST_CAPTURE(x) (static_cast<void>(0))
02635 #define DOCTEST_ADD_MESSAGE_AT(file, line, ...) (static_cast<void>(0))
02636 #define DOCTEST_ADD_FAIL_CHECK_AT(file, line, ...) (static_cast<void>(0))
02637 #define DOCTEST_ADD_FAIL_AT(file, line, ...) (static_cast<void>(0))
02638 #define DOCTEST_MESSAGE(...) (static_cast<void>(0))
02639 #define DOCTEST_FAIL_CHECK(...) (static_cast<void>(0))
02640 #define DOCTEST_FAIL(...) (static_cast<void>(0))
02641
02642 #if defined(DOCTEST_CONFIG_EVALUATE_ASSERTS_EVEN_WHEN_DISABLED) \
02643     && defined(DOCTEST_CONFIG_ASSERTS_RETURN_VALUES)
02644
02645 #define DOCTEST_WARN(...) [&] { return __VA_ARGS__; }()
02646 #define DOCTEST_CHECK(...) [&] { return __VA_ARGS__; }()
02647 #define DOCTEST_REQUIRE(...) [&] { return __VA_ARGS__; }()
02648 #define DOCTEST_WARN_FALSE(...) [&] { return !(__VA_ARGS__); }()
02649 #define DOCTEST_CHECK_FALSE(...) [&] { return !(__VA_ARGS__); }()
02650 #define DOCTEST_REQUIRE_FALSE(...) [&] { return !(__VA_ARGS__); }()
02651
02652 #define DOCTEST_WARN_MESSAGE(cond, ...) [&] { return cond; }()
02653 #define DOCTEST_CHECK_MESSAGE(cond, ...) [&] { return cond; }()
02654 #define DOCTEST_REQUIRE_MESSAGE(cond, ...) [&] { return cond; }()
02655 #define DOCTEST_WARN_FALSE_MESSAGE(cond, ...) [&] { return !(cond); }()
02656 #define DOCTEST_CHECK_FALSE_MESSAGE(cond, ...) [&] { return !(cond); }()
02657 #define DOCTEST_REQUIRE_FALSE_MESSAGE(cond, ...) [&] { return !(cond); }()
02658
02659 namespace doctest {
02660     namespace detail {
02661         #define DOCTEST_RELATIONAL_OP(name, op) \
02662             template <typename L, typename R> \
02663             bool name(const DOCTEST_REF_WRAP(L) lhs, const DOCTEST_REF_WRAP(R) rhs) { return lhs op rhs; }
02664
02665         DOCTEST_RELATIONAL_OP(eq, ==)
02666         DOCTEST_RELATIONAL_OP(ne, !=)
02667         DOCTEST_RELATIONAL_OP(lt, <)
02668         DOCTEST_RELATIONAL_OP(gt, >)
02669         DOCTEST_RELATIONAL_OP(le, <=)
02670         DOCTEST_RELATIONAL_OP(ge, >=)
02671     } // namespace detail
02672 } // namespace doctest
02673
02674 #define DOCTEST_WARN_EQ(...) [&] { return doctest::detail::eq(__VA_ARGS__); }()
02675 #define DOCTEST_CHECK_EQ(...) [&] { return doctest::detail::eq(__VA_ARGS__); }()
02676 #define DOCTEST_REQUIRE_EQ(...) [&] { return doctest::detail::eq(__VA_ARGS__); }()
02677 #define DOCTEST_WARN_NE(...) [&] { return doctest::detail::ne(__VA_ARGS__); }()
02678 #define DOCTEST_CHECK_NE(...) [&] { return doctest::detail::ne(__VA_ARGS__); }()
02679 #define DOCTEST_REQUIRE_NE(...) [&] { return doctest::detail::ne(__VA_ARGS__); }()
02680 #define DOCTEST_WARN_LT(...) [&] { return doctest::detail::lt(__VA_ARGS__); }()
02681 #define DOCTEST_CHECK_LT(...) [&] { return doctest::detail::lt(__VA_ARGS__); }()
02682 #define DOCTEST_REQUIRE_LT(...) [&] { return doctest::detail::lt(__VA_ARGS__); }()
02683 #define DOCTEST_WARN_GT(...) [&] { return doctest::detail::gt(__VA_ARGS__); }()
02684 #define DOCTEST_CHECK_GT(...) [&] { return doctest::detail::gt(__VA_ARGS__); }()
02685 #define DOCTEST_REQUIRE_GT(...) [&] { return doctest::detail::gt(__VA_ARGS__); }()
02686 #define DOCTEST_WARN_LE(...) [&] { return doctest::detail::le(__VA_ARGS__); }()
02687 #define DOCTEST_CHECK_LE(...) [&] { return doctest::detail::le(__VA_ARGS__); }()
02688 #define DOCTEST_REQUIRE_LE(...) [&] { return doctest::detail::le(__VA_ARGS__); }()
02689 #define DOCTEST_WARN_GE(...) [&] { return doctest::detail::ge(__VA_ARGS__); }()
02690 #define DOCTEST_CHECK_GE(...) [&] { return doctest::detail::ge(__VA_ARGS__); }()
02691 #define DOCTEST_REQUIRE_GE(...) [&] { return doctest::detail::ge(__VA_ARGS__); }()
02692 #define DOCTEST_WARN_UNARY(...) [&] { return __VA_ARGS__; }()
02693 #define DOCTEST_CHECK_UNARY(...) [&] { return __VA_ARGS__; }()
02694 #define DOCTEST_REQUIRE_UNARY(...) [&] { return __VA_ARGS__; }()
02695 #define DOCTEST_WARN_UNARY_FALSE(...) [&] { return !(__VA_ARGS__); }()
02696 #define DOCTEST_CHECK_UNARY_FALSE(...) [&] { return !(__VA_ARGS__); }()
02697 #define DOCTEST_REQUIRE_UNARY_FALSE(...) [&] { return !(__VA_ARGS__); }()
02698
02699 #ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
02700
02701 #define DOCTEST_WARN_THROWS_WITH(expr, with, ...) [] { static_assert(false, "Exception translation is not available when doctest is disabled."); return false; }()
02702 #define DOCTEST_CHECK_THROWS_WITH(expr, with, ...) DOCTEST_WARN_THROWS_WITH(„)
02703 #define DOCTEST_REQUIRE_THROWS_WITH(expr, with, ...) DOCTEST_WARN_THROWS_WITH(„)
02704 #define DOCTEST_WARN_THROWS_WITH_AS(expr, with, ex, ...) DOCTEST_WARN_THROWS_WITH(„)
02705 #define DOCTEST_CHECK_THROWS_WITH_AS(expr, with, ex, ...) DOCTEST_WARN_THROWS_WITH(„)
02706 #define DOCTEST_REQUIRE_THROWS_WITH_AS(expr, with, ex, ...) DOCTEST_WARN_THROWS_WITH(„)
02707
02708 #define DOCTEST_WARN_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_WARN_THROWS_WITH(„)
02709 #define DOCTEST_CHECK_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_WARN_THROWS_WITH(„)
02710 #define DOCTEST_REQUIRE_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_WARN_THROWS_WITH(„)
02711 #define DOCTEST_WARN_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_WARN_THROWS_WITH(„)
02712 #define DOCTEST_CHECK_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_WARN_THROWS_WITH(„)
02713 #define DOCTEST_REQUIRE_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_WARN_THROWS_WITH(„)
02714
02715 #define DOCTEST_WARN_THROWS(...) [&] { try { __VA_ARGS__; return false; } catch (...) { return true; } }()
02716 #define DOCTEST_CHECK_THROWS(...) [&] { try { __VA_ARGS__; return false; } catch (...) { return true; } }()

```

```

    } }()
02717 #define DOCTEST_REQUIRE_THROWS(...) [&] { try { __VA_ARGS__; return false; } catch (...) { return
true; } }()
02718 #define DOCTEST_WARN_THROWS_AS(expr, ...) [&] { try { expr; } catch (__VA_ARGS__) { return true; }
catch (...) { } return false; }()
02719 #define DOCTEST_CHECK_THROWS_AS(expr, ...) [&] { try { expr; } catch (__VA_ARGS__) { return true; }
catch (...) { } return false; }()
02720 #define DOCTEST_REQUIRE_THROWS_AS(expr, ...) [&] { try { expr; } catch (__VA_ARGS__) { return true; }
catch (...) { } return false; }()
02721 #define DOCTEST_WARN_NOTHROW(...) [&] { try { __VA_ARGS__; return true; } catch (...) { return false;
} }()
02722 #define DOCTEST_CHECK_NOTHROW(...) [&] { try { __VA_ARGS__; return true; } catch (...) { return false;
} }()
02723 #define DOCTEST_REQUIRE_NOTHROW(...) [&] { try { __VA_ARGS__; return true; } catch (...) { return
false; } }()
02724
02725 #define DOCTEST_WARN_THROWS_MESSAGE(expr, ...) [&] { try { __VA_ARGS__; return false; } catch (...) {
return true; } }()
02726 #define DOCTEST_CHECK_THROWS_MESSAGE(expr, ...) [&] { try { __VA_ARGS__; return false; } catch (...) {
return true; } }()
02727 #define DOCTEST_REQUIRE_THROWS_MESSAGE(expr, ...) [&] { try { __VA_ARGS__; return false; } catch (...)
{ return true; } }()
02728 #define DOCTEST_WARN_THROWS_AS_MESSAGE(expr, ex, ...) [&] { try { expr; } catch (__VA_ARGS__) { return
true; } catch (...) { } return false; }()
02729 #define DOCTEST_CHECK_THROWS_AS_MESSAGE(expr, ex, ...) [&] { try { expr; } catch (__VA_ARGS__) {
return true; } catch (...) { } return false; }()
02730 #define DOCTEST_REQUIRE_THROWS_AS_MESSAGE(expr, ex, ...) [&] { try { expr; } catch (__VA_ARGS__) {
return true; } catch (...) { } return false; }()
02731 #define DOCTEST_WARN_NOTHROW_MESSAGE(expr, ...) [&] { try { __VA_ARGS__; return true; } catch (...) {
return false; } }()
02732 #define DOCTEST_CHECK_NOTHROW_MESSAGE(expr, ...) [&] { try { __VA_ARGS__; return true; } catch (...) {
return false; } }()
02733 #define DOCTEST_REQUIRE_NOTHROW_MESSAGE(expr, ...) [&] { try { __VA_ARGS__; return true; } catch (...)
{ return false; } }()
02734
02735 #endif // DOCTEST_CONFIG_NO_EXCEPTIONS
02736
02737 #else // DOCTEST_CONFIG_EVALUATE_ASSERTS_EVEN_WHEN_DISABLED
02738
02739 #define DOCTEST_WARN(...) DOCTEST_FUNC_EMPTY
02740 #define DOCTEST_CHECK(...) DOCTEST_FUNC_EMPTY
02741 #define DOCTEST_REQUIRE(...) DOCTEST_FUNC_EMPTY
02742 #define DOCTEST_WARN_FALSE(...) DOCTEST_FUNC_EMPTY
02743 #define DOCTEST_CHECK_FALSE(...) DOCTEST_FUNC_EMPTY
02744 #define DOCTEST_REQUIRE_FALSE(...) DOCTEST_FUNC_EMPTY
02745
02746 #define DOCTEST_WARN_MESSAGE(cond, ...) DOCTEST_FUNC_EMPTY
02747 #define DOCTEST_CHECK_MESSAGE(cond, ...) DOCTEST_FUNC_EMPTY
02748 #define DOCTEST_REQUIRE_MESSAGE(cond, ...) DOCTEST_FUNC_EMPTY
02749 #define DOCTEST_WARN_FALSE_MESSAGE(cond, ...) DOCTEST_FUNC_EMPTY
02750 #define DOCTEST_CHECK_FALSE_MESSAGE(cond, ...) DOCTEST_FUNC_EMPTY
02751 #define DOCTEST_REQUIRE_FALSE_MESSAGE(cond, ...) DOCTEST_FUNC_EMPTY
02752
02753 #define DOCTEST_WARN_EQ(...) DOCTEST_FUNC_EMPTY
02754 #define DOCTEST_CHECK_EQ(...) DOCTEST_FUNC_EMPTY
02755 #define DOCTEST_REQUIRE_EQ(...) DOCTEST_FUNC_EMPTY
02756 #define DOCTEST_WARN_NE(...) DOCTEST_FUNC_EMPTY
02757 #define DOCTEST_CHECK_NE(...) DOCTEST_FUNC_EMPTY
02758 #define DOCTEST_REQUIRE_NE(...) DOCTEST_FUNC_EMPTY
02759 #define DOCTEST_WARN_GT(...) DOCTEST_FUNC_EMPTY
02760 #define DOCTEST_CHECK_GT(...) DOCTEST_FUNC_EMPTY
02761 #define DOCTEST_REQUIRE_GT(...) DOCTEST_FUNC_EMPTY
02762 #define DOCTEST_WARN_LT(...) DOCTEST_FUNC_EMPTY
02763 #define DOCTEST_CHECK_LT(...) DOCTEST_FUNC_EMPTY
02764 #define DOCTEST_REQUIRE_LT(...) DOCTEST_FUNC_EMPTY
02765 #define DOCTEST_WARN_GE(...) DOCTEST_FUNC_EMPTY
02766 #define DOCTEST_CHECK_GE(...) DOCTEST_FUNC_EMPTY
02767 #define DOCTEST_REQUIRE_GE(...) DOCTEST_FUNC_EMPTY
02768 #define DOCTEST_WARN_LE(...) DOCTEST_FUNC_EMPTY
02769 #define DOCTEST_CHECK_LE(...) DOCTEST_FUNC_EMPTY
02770 #define DOCTEST_REQUIRE_LE(...) DOCTEST_FUNC_EMPTY
02771
02772 #define DOCTEST_WARN_UNARY(...) DOCTEST_FUNC_EMPTY
02773 #define DOCTEST_CHECK_UNARY(...) DOCTEST_FUNC_EMPTY
02774 #define DOCTEST_REQUIRE_UNARY(...) DOCTEST_FUNC_EMPTY
02775 #define DOCTEST_WARN_UNARY_FALSE(...) DOCTEST_FUNC_EMPTY
02776 #define DOCTEST_CHECK_UNARY_FALSE(...) DOCTEST_FUNC_EMPTY
02777 #define DOCTEST_REQUIRE_UNARY_FALSE(...) DOCTEST_FUNC_EMPTY
02778
02779 #ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
02780
02781 #define DOCTEST_WARN_THROWS(...) DOCTEST_FUNC_EMPTY
02782 #define DOCTEST_CHECK_THROWS(...) DOCTEST_FUNC_EMPTY
02783 #define DOCTEST_REQUIRE_THROWS(...) DOCTEST_FUNC_EMPTY
02784 #define DOCTEST_WARN_THROWS_AS(expr, ...) DOCTEST_FUNC_EMPTY
02785 #define DOCTEST_CHECK_THROWS_AS(expr, ...) DOCTEST_FUNC_EMPTY
02786 #define DOCTEST_REQUIRE_THROWS_AS(expr, ...) DOCTEST_FUNC_EMPTY

```

```
02787 #define DOCTEST_WARN_THROWS_WITH(expr, ...) DOCTEST_FUNC_EMPTY
02788 #define DOCTEST_CHECK_THROWS_WITH(expr, ...) DOCTEST_FUNC_EMPTY
02789 #define DOCTEST_REQUIRE_THROWS_WITH(expr, ...) DOCTEST_FUNC_EMPTY
02790 #define DOCTEST_WARN_THROWS_WITH_AS(expr, with, ...) DOCTEST_FUNC_EMPTY
02791 #define DOCTEST_CHECK_THROWS_WITH_AS(expr, with, ...) DOCTEST_FUNC_EMPTY
02792 #define DOCTEST_REQUIRE_THROWS_WITH_AS(expr, with, ...) DOCTEST_FUNC_EMPTY
02793 #define DOCTEST_WARN_NOTHROW(...) DOCTEST_FUNC_EMPTY
02794 #define DOCTEST_CHECK_NOTHROW(...) DOCTEST_FUNC_EMPTY
02795 #define DOCTEST_REQUIRE_NOTHROW(...) DOCTEST_FUNC_EMPTY
02796
02797 #define DOCTEST_WARN_THROWS_MESSAGE(expr, ...) DOCTEST_FUNC_EMPTY
02798 #define DOCTEST_CHECK_THROWS_MESSAGE(expr, ...) DOCTEST_FUNC_EMPTY
02799 #define DOCTEST_REQUIRE_THROWS_MESSAGE(expr, ...) DOCTEST_FUNC_EMPTY
02800 #define DOCTEST_WARN_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_FUNC_EMPTY
02801 #define DOCTEST_CHECK_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_FUNC_EMPTY
02802 #define DOCTEST_REQUIRE_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_FUNC_EMPTY
02803 #define DOCTEST_WARN_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_FUNC_EMPTY
02804 #define DOCTEST_CHECK_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_FUNC_EMPTY
02805 #define DOCTEST_REQUIRE_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_FUNC_EMPTY
02806 #define DOCTEST_WARN_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_FUNC_EMPTY
02807 #define DOCTEST_CHECK_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_FUNC_EMPTY
02808 #define DOCTEST_REQUIRE_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_FUNC_EMPTY
02809 #define DOCTEST_WARN_NOTHROW_MESSAGE(expr, ...) DOCTEST_FUNC_EMPTY
02810 #define DOCTEST_CHECK_NOTHROW_MESSAGE(expr, ...) DOCTEST_FUNC_EMPTY
02811 #define DOCTEST_REQUIRE_NOTHROW_MESSAGE(expr, ...) DOCTEST_FUNC_EMPTY
02812
02813 #endif // DOCTEST_CONFIG_NO_EXCEPTIONS
02814
02815 #endif // DOCTEST_CONFIG_EVALUATE_ASSERTS_EVEN_WHEN_DISABLED
02816
02817 #endif // DOCTEST_CONFIG_DISABLE
02818
02819 #ifdef DOCTEST_CONFIG_NO_EXCEPTIONS
02820
02821 #ifdef DOCTEST_CONFIG_NO_EXCEPTIONS_BUT_WITH_ALL_ASSERTS
02822 #define DOCTEST_EXCEPTION_EMPTY_FUNC DOCTEST_FUNC_EMPTY
02823 #else // DOCTEST_CONFIG_NO_EXCEPTIONS_BUT_WITH_ALL_ASSERTS
02824 #define DOCTEST_EXCEPTION_EMPTY_FUNC [] { static_assert(false, "Exceptions are disabled! " \
02825     "Use DOCTEST_CONFIG_NO_EXCEPTIONS_BUT_WITH_ALL_ASSERTS if you want to compile with exceptions \
02826     disabled."); return false; }()
02827 #undef DOCTEST_REQUIRE
02828 #undef DOCTEST_REQUIRE_FALSE
02829 #undef DOCTEST_REQUIRE_MESSAGE
02830 #undef DOCTEST_REQUIRE_FALSE_MESSAGE
02831 #undef DOCTEST_REQUIRE_EQ
02832 #undef DOCTEST_REQUIRE_NE
02833 #undef DOCTEST_REQUIRE_GT
02834 #undef DOCTEST_REQUIRE_LT
02835 #undef DOCTEST_REQUIRE_GE
02836 #undef DOCTEST_REQUIRE_LE
02837 #undef DOCTEST_REQUIRE_UNARY
02838 #undef DOCTEST_REQUIRE_UNARY_FALSE
02839
02840 #define DOCTEST_REQUIRE DOCTEST_EXCEPTION_EMPTY_FUNC
02841 #define DOCTEST_REQUIRE_FALSE DOCTEST_EXCEPTION_EMPTY_FUNC
02842 #define DOCTEST_REQUIRE_MESSAGE DOCTEST_EXCEPTION_EMPTY_FUNC
02843 #define DOCTEST_REQUIRE_FALSE_MESSAGE DOCTEST_EXCEPTION_EMPTY_FUNC
02844 #define DOCTEST_REQUIRE_EQ DOCTEST_EXCEPTION_EMPTY_FUNC
02845 #define DOCTEST_REQUIRE_NE DOCTEST_EXCEPTION_EMPTY_FUNC
02846 #define DOCTEST_REQUIRE_GT DOCTEST_EXCEPTION_EMPTY_FUNC
02847 #define DOCTEST_REQUIRE_LT DOCTEST_EXCEPTION_EMPTY_FUNC
02848 #define DOCTEST_REQUIRE_GE DOCTEST_EXCEPTION_EMPTY_FUNC
02849 #define DOCTEST_REQUIRE_LE DOCTEST_EXCEPTION_EMPTY_FUNC
02850 #define DOCTEST_REQUIRE_UNARY DOCTEST_EXCEPTION_EMPTY_FUNC
02851 #define DOCTEST_REQUIRE_UNARY_FALSE DOCTEST_EXCEPTION_EMPTY_FUNC
02852
02853 #endif // DOCTEST_CONFIG_NO_EXCEPTIONS_BUT_WITH_ALL_ASSERTS
02854
02855 #define DOCTEST_WARN_THROWS(...) DOCTEST_EXCEPTION_EMPTY_FUNC
02856 #define DOCTEST_CHECK_THROWS(...) DOCTEST_EXCEPTION_EMPTY_FUNC
02857 #define DOCTEST_REQUIRE_THROWS(...) DOCTEST_EXCEPTION_EMPTY_FUNC
02858 #define DOCTEST_WARN_THROWS_AS(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02859 #define DOCTEST_CHECK_THROWS_AS(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02860 #define DOCTEST_REQUIRE_THROWS_AS(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02861 #define DOCTEST_WARN_THROWS_WITH(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02862 #define DOCTEST_CHECK_THROWS_WITH(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02863 #define DOCTEST_REQUIRE_THROWS_WITH(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02864 #define DOCTEST_WARN_THROWS_WITH_AS(expr, with, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02865 #define DOCTEST_CHECK_THROWS_WITH_AS(expr, with, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02866 #define DOCTEST_REQUIRE_THROWS_WITH_AS(expr, with, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02867 #define DOCTEST_WARN_NOTHROW(...) DOCTEST_EXCEPTION_EMPTY_FUNC
02868 #define DOCTEST_CHECK_NOTHROW(...) DOCTEST_EXCEPTION_EMPTY_FUNC
02869 #define DOCTEST_REQUIRE_NOTHROW(...) DOCTEST_EXCEPTION_EMPTY_FUNC
02870
02871 #define DOCTEST_WARN_THROWS_MESSAGE(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02872 #define DOCTEST_CHECK_THROWS_MESSAGE(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
```

```

02873 #define DOCTEST_REQUIRE_THROWS_MESSAGE(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02874 #define DOCTEST_WARN_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02875 #define DOCTEST_CHECK_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02876 #define DOCTEST_REQUIRE_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02877 #define DOCTEST_WARN_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02878 #define DOCTEST_CHECK_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02879 #define DOCTEST_REQUIRE_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02880 #define DOCTEST_WARN_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02881 #define DOCTEST_CHECK_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02882 #define DOCTEST_REQUIRE_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02883 #define DOCTEST_WARN_NOTHROW_MESSAGE(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02884 #define DOCTEST_CHECK_NOTHROW_MESSAGE(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02885 #define DOCTEST_REQUIRE_NOTHROW_MESSAGE(expr, ...) DOCTEST_EXCEPTION_EMPTY_FUNC
02886
02887 #endif // DOCTEST_CONFIG_NO_EXCEPTIONS
02888
02889 // clang-format off
02890 // KEPT FOR BACKWARDS COMPATIBILITY - FORWARDING TO THE RIGHT MACROS
02891 #define DOCTEST_FAST_WARN_EQ DOCTEST_WARN_EQ
02892 #define DOCTEST_FAST_CHECK_EQ DOCTEST_CHECK_EQ
02893 #define DOCTEST_FAST_REQUIRE_EQ DOCTEST_REQUIRE_EQ
02894 #define DOCTEST_FAST_WARN_NE DOCTEST_WARN_NE
02895 #define DOCTEST_FAST_CHECK_NE DOCTEST_CHECK_NE
02896 #define DOCTEST_FAST_REQUIRE_NE DOCTEST_REQUIRE_NE
02897 #define DOCTEST_FAST_WARN_GT DOCTEST_WARN_GT
02898 #define DOCTEST_FAST_CHECK_GT DOCTEST_CHECK_GT
02899 #define DOCTEST_FAST_REQUIRE_GT DOCTEST_REQUIRE_GT
02900 #define DOCTEST_FAST_WARN_LT DOCTEST_WARN_LT
02901 #define DOCTEST_FAST_CHECK_LT DOCTEST_CHECK_LT
02902 #define DOCTEST_FAST_REQUIRE_LT DOCTEST_REQUIRE_LT
02903 #define DOCTEST_FAST_WARN_GE DOCTEST_WARN_GE
02904 #define DOCTEST_FAST_CHECK_GE DOCTEST_CHECK_GE
02905 #define DOCTEST_FAST_REQUIRE_GE DOCTEST_REQUIRE_GE
02906 #define DOCTEST_FAST_WARN_LE DOCTEST_WARN_LE
02907 #define DOCTEST_FAST_CHECK_LE DOCTEST_CHECK_LE
02908 #define DOCTEST_FAST_REQUIRE_LE DOCTEST_REQUIRE_LE
02909
02910 #define DOCTEST_FAST_WARN_UNARY DOCTEST_WARN_UNARY
02911 #define DOCTEST_FAST_CHECK_UNARY DOCTEST_CHECK_UNARY
02912 #define DOCTEST_FAST_REQUIRE_UNARY DOCTEST_REQUIRE_UNARY
02913 #define DOCTEST_FAST_WARN_UNARY_FALSE DOCTEST_WARN_UNARY_FALSE
02914 #define DOCTEST_FAST_CHECK_UNARY_FALSE DOCTEST_CHECK_UNARY_FALSE
02915 #define DOCTEST_FAST_REQUIRE_UNARY_FALSE DOCTEST_REQUIRE_UNARY_FALSE
02916
02917 #define DOCTEST_TEST_CASE_TEMPLATE_INSTANTIATE(id, ...)
DOCTEST_TEST_CASE_TEMPLATE_INVOKE(id, __VA_ARGS__)
02918 // clang-format on
02919
02920 // BDD style macros
02921 // clang-format off
02922 #define DOCTEST_SCENARIO(name) DOCTEST_TEST_CASE(" Scenario: " name)
02923 #define DOCTEST_SCENARIO_CLASS(name) DOCTEST_TEST_CASE_CLASS(" Scenario: " name)
02924 #define DOCTEST_SCENARIO_TEMPLATE(name, T, ...) DOCTEST_TEST_CASE_TEMPLATE(" Scenario: " name, T,
__VA_ARGS__)
02925 #define DOCTEST_SCENARIO_TEMPLATE_DEFINE(name, T, id) DOCTEST_TEST_CASE_TEMPLATE_DEFINE(" Scenario: "
name, T, id)
02926
02927 #define DOCTEST_GIVEN(name) DOCTEST_SUBCASE(" Given: " name)
02928 #define DOCTEST_WHEN(name) DOCTEST_SUBCASE(" When: " name)
02929 #define DOCTEST_AND_WHEN(name) DOCTEST_SUBCASE("And when: " name)
02930 #define DOCTEST_THEN(name) DOCTEST_SUBCASE(" Then: " name)
02931 #define DOCTEST_AND_THEN(name) DOCTEST_SUBCASE(" And: " name)
02932 // clang-format on
02933
02934 // == SHORT VERSIONS OF THE MACROS
02935 #ifndef DOCTEST_CONFIG_NO_SHORT_MACRO_NAMES
02936
02937 #define TEST_CASE(name) DOCTEST_TEST_CASE(name)
02938 #define TEST_CASE_CLASS(name) DOCTEST_TEST_CASE_CLASS(name)
02939 #define TEST_CASE_FIXTURE(x, name) DOCTEST_TEST_CASE_FIXTURE(x, name)
02940 #define TYPE_TO_STRING_AS(str, ...) DOCTEST_TYPE_TO_STRING_AS(str, __VA_ARGS__)
02941 #define TYPE_TO_STRING(...) DOCTEST_TYPE_TO_STRING(__VA_ARGS__)
02942 #define TEST_CASE_TEMPLATE(name, T, ...) DOCTEST_TEST_CASE_TEMPLATE(name, T, __VA_ARGS__)
02943 #define TEST_CASE_TEMPLATE_DEFINE(name, T, id) DOCTEST_TEST_CASE_TEMPLATE_DEFINE(name, T, id)
02944 #define TEST_CASE_TEMPLATE_INVOKE(id, ...) DOCTEST_TEST_CASE_TEMPLATE_INVOKE(id, __VA_ARGS__)
02945 #define TEST_CASE_TEMPLATE_APPLY(id, ...) DOCTEST_TEST_CASE_TEMPLATE_APPLY(id, __VA_ARGS__)
02946 #define SUBCASE(name) DOCTEST_SUBCASE(name)
02947 #define TEST_SUITE(decorators) DOCTEST_TEST_SUITE(decorators)
02948 #define TEST_SUITE_BEGIN(name) DOCTEST_TEST_SUITE_BEGIN(name)
02949 #define TEST_SUITE_END DOCTEST_TEST_SUITE_END
02950 #define REGISTER_EXCEPTION_TRANSLATOR(signature) DOCTEST_REGISTER_EXCEPTION_TRANSLATOR(signature)
02951 #define REGISTER_REPORTER(name, priority, reporter) DOCTEST_REGISTER_REPORTER(name, priority,
reporter)
02952 #define REGISTER_LISTENER(name, priority, reporter) DOCTEST_REGISTER_LISTENER(name, priority,
reporter)
02953 #define INFO(...) DOCTEST_INFO(__VA_ARGS__)
02954 #define CAPTURE(x) DOCTEST_CAPTURE(x)

```



```
02955 #define ADD_MESSAGE_AT(file, line, ...) DOCTEST_ADD_MESSAGE_AT(file, line, __VA_ARGS__)
02956 #define ADD_FAIL_CHECK_AT(file, line, ...) DOCTEST_ADD_FAIL_CHECK_AT(file, line, __VA_ARGS__)
02957 #define ADD_FAIL_AT(file, line, ...) DOCTEST_ADD_FAIL_AT(file, line, __VA_ARGS__)
02958 #define MESSAGE(...) DOCTEST_MESSAGE(__VA_ARGS__)
02959 #define FAIL_CHECK(...) DOCTEST_FAIL_CHECK(__VA_ARGS__)
02960 #define FAIL(...) DOCTEST_FAIL(__VA_ARGS__)
02961 #define TO_LVALUE(...) DOCTEST_TO_LVALUE(__VA_ARGS__)
02962
02963 #define WARN(...) DOCTEST_WARN(__VA_ARGS__)
02964 #define WARN_FALSE(...) DOCTEST_WARN_FALSE(__VA_ARGS__)
02965 #define WARN_THROWS(...) DOCTEST_WARN_THROWS(__VA_ARGS__)
02966 #define WARN_THROWS_AS(expr, ...) DOCTEST_WARN_THROWS_AS(expr, __VA_ARGS__)
02967 #define WARN_THROWS_WITH(expr, ...) DOCTEST_WARN_THROWS_WITH(expr, __VA_ARGS__)
02968 #define WARN_THROWS_WITH_AS(expr, with, ...) DOCTEST_WARN_THROWS_WITH_AS(expr, with, __VA_ARGS__)
02969 #define WARN_NOTHROW(...) DOCTEST_WARN_NOTHROW(__VA_ARGS__)
02970 #define CHECK(...) DOCTEST_CHECK(__VA_ARGS__)
02971 #define CHECK_FALSE(...) DOCTEST_CHECK_FALSE(__VA_ARGS__)
02972 #define CHECK_THROWS(...) DOCTEST_CHECK_THROWS(__VA_ARGS__)
02973 #define CHECK_THROWS_AS(expr, ...) DOCTEST_CHECK_THROWS_AS(expr, __VA_ARGS__)
02974 #define CHECK_THROWS_WITH(expr, ...) DOCTEST_CHECK_THROWS_WITH(expr, __VA_ARGS__)
02975 #define CHECK_THROWS_WITH_AS(expr, with, ...) DOCTEST_CHECK_THROWS_WITH_AS(expr, with, __VA_ARGS__)
02976 #define CHECK_NOTHROW(...) DOCTEST_CHECK_NOTHROW(__VA_ARGS__)
02977 #define REQUIRE(...) DOCTEST_REQUIRE(__VA_ARGS__)
02978 #define REQUIRE_FALSE(...) DOCTEST_REQUIRE_FALSE(__VA_ARGS__)
02979 #define REQUIRE_THROWS(...) DOCTEST_REQUIRE_THROWS(__VA_ARGS__)
02980 #define REQUIRE_THROWS_AS(expr, ...) DOCTEST_REQUIRE_THROWS_AS(expr, __VA_ARGS__)
02981 #define REQUIRE_THROWS_WITH(expr, ...) DOCTEST_REQUIRE_THROWS_WITH(expr, __VA_ARGS__)
02982 #define REQUIRE_THROWS_WITH_AS(expr, with, ...) DOCTEST_REQUIRE_THROWS_WITH_AS(expr, with,
__VA_ARGS__)
02983 #define REQUIRE_NOTHROW(...) DOCTEST_REQUIRE_NOTHROW(__VA_ARGS__)
02984
02985 #define WARN_MESSAGE(cond, ...) DOCTEST_WARN_MESSAGE(cond, __VA_ARGS__)
02986 #define WARN_FALSE_MESSAGE(cond, ...) DOCTEST_WARN_FALSE_MESSAGE(cond, __VA_ARGS__)
02987 #define WARN_THROWS_MESSAGE(expr, ...) DOCTEST_WARN_THROWS_MESSAGE(expr, __VA_ARGS__)
02988 #define WARN_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_WARN_THROWS_AS_MESSAGE(expr, ex, __VA_ARGS__)
02989 #define WARN_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_WARN_THROWS_WITH_MESSAGE(expr, with,
__VA_ARGS__)
02990 #define WARN_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_WARN_THROWS_WITH_AS_MESSAGE(expr,
with, ex, __VA_ARGS__)
02991 #define WARN_NOTHROW_MESSAGE(expr, ...) DOCTEST_WARN_NOTHROW_MESSAGE(expr, __VA_ARGS__)
02992 #define CHECK_MESSAGE(cond, ...) DOCTEST_CHECK_MESSAGE(cond, __VA_ARGS__)
02993 #define CHECK_FALSE_MESSAGE(cond, ...) DOCTEST_CHECK_FALSE_MESSAGE(cond, __VA_ARGS__)
02994 #define CHECK_THROWS_MESSAGE(expr, ...) DOCTEST_CHECK_THROWS_MESSAGE(expr, __VA_ARGS__)
02995 #define CHECK_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_CHECK_THROWS_AS_MESSAGE(expr, ex, __VA_ARGS__)
02996 #define CHECK_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_CHECK_THROWS_WITH_MESSAGE(expr, with,
__VA_ARGS__)
02997 #define CHECK_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...) DOCTEST_CHECK_THROWS_WITH_AS_MESSAGE(expr,
with, ex, __VA_ARGS__)
02998 #define CHECK_NOTHROW_MESSAGE(expr, ...) DOCTEST_CHECK_NOTHROW_MESSAGE(expr, __VA_ARGS__)
02999 #define REQUIRE_MESSAGE(cond, ...) DOCTEST_REQUIRE_MESSAGE(cond, __VA_ARGS__)
03000 #define REQUIRE_FALSE_MESSAGE(cond, ...) DOCTEST_REQUIRE_FALSE_MESSAGE(cond, __VA_ARGS__)
03001 #define REQUIRE_THROWS_MESSAGE(expr, ...) DOCTEST_REQUIRE_THROWS_MESSAGE(expr, __VA_ARGS__)
03002 #define REQUIRE_THROWS_AS_MESSAGE(expr, ex, ...) DOCTEST_REQUIRE_THROWS_AS_MESSAGE(expr, ex,
__VA_ARGS__)
03003 #define REQUIRE_THROWS_WITH_MESSAGE(expr, with, ...) DOCTEST_REQUIRE_THROWS_WITH_MESSAGE(expr, with,
__VA_ARGS__)
03004 #define REQUIRE_THROWS_WITH_AS_MESSAGE(expr, with, ex, ...)
DOCTEST_REQUIRE_THROWS_WITH_AS_MESSAGE(expr, with, ex, __VA_ARGS__)
03005 #define REQUIRE_NOTHROW_MESSAGE(expr, ...) DOCTEST_REQUIRE_NOTHROW_MESSAGE(expr, __VA_ARGS__)
03006
03007 #define SCENARIO(name) DOCTEST_SCENARIO(name)
03008 #define SCENARIO_CLASS(name) DOCTEST_SCENARIO_CLASS(name)
03009 #define SCENARIO_TEMPLATE(name, T, ...) DOCTEST_SCENARIO_TEMPLATE(name, T, __VA_ARGS__)
03010 #define SCENARIO_TEMPLATE_DEFINE(name, T, id) DOCTEST_SCENARIO_TEMPLATE_DEFINE(name, T, id)
03011 #define GIVEN(name) DOCTEST_GIVEN(name)
03012 #define WHEN(name) DOCTEST_WHEN(name)
03013 #define AND_WHEN(name) DOCTEST_AND_WHEN(name)
03014 #define THEN(name) DOCTEST_THEN(name)
03015 #define AND_THEN(name) DOCTEST_AND_THEN(name)
03016
03017 #define WARN_EQ(...) DOCTEST_WARN_EQ(__VA_ARGS__)
03018 #define CHECK_EQ(...) DOCTEST_CHECK_EQ(__VA_ARGS__)
03019 #define REQUIRE_EQ(...) DOCTEST_REQUIRE_EQ(__VA_ARGS__)
03020 #define WARN_NE(...) DOCTEST_WARN_NE(__VA_ARGS__)
03021 #define CHECK_NE(...) DOCTEST_CHECK_NE(__VA_ARGS__)
03022 #define REQUIRE_NE(...) DOCTEST_REQUIRE_NE(__VA_ARGS__)
03023 #define WARN_GT(...) DOCTEST_WARN_GT(__VA_ARGS__)
03024 #define CHECK_GT(...) DOCTEST_CHECK_GT(__VA_ARGS__)
03025 #define REQUIRE_GT(...) DOCTEST_REQUIRE_GT(__VA_ARGS__)
03026 #define WARN_LT(...) DOCTEST_WARN_LT(__VA_ARGS__)
03027 #define CHECK_LT(...) DOCTEST_CHECK_LT(__VA_ARGS__)
03028 #define REQUIRE_LT(...) DOCTEST_REQUIRE_LT(__VA_ARGS__)
03029 #define WARN_GE(...) DOCTEST_WARN_GE(__VA_ARGS__)
03030 #define CHECK_GE(...) DOCTEST_CHECK_GE(__VA_ARGS__)
03031 #define REQUIRE_GE(...) DOCTEST_REQUIRE_GE(__VA_ARGS__)
03032 #define WARN_LE(...) DOCTEST_WARN_LE(__VA_ARGS__)
03033 #define CHECK_LE(...) DOCTEST_CHECK_LE(__VA_ARGS__)
```

```

03034 #define REQUIRE_LE(...) DOCTEST_REQUIRE_LE(__VA_ARGS__)
03035 #define WARN_UNARY(...) DOCTEST_WARN_UNARY(__VA_ARGS__)
03036 #define CHECK_UNARY(...) DOCTEST_CHECK_UNARY(__VA_ARGS__)
03037 #define REQUIRE_UNARY(...) DOCTEST_REQUIRE_UNARY(__VA_ARGS__)
03038 #define WARN_UNARY_FALSE(...) DOCTEST_WARN_UNARY_FALSE(__VA_ARGS__)
03039 #define CHECK_UNARY_FALSE(...) DOCTEST_CHECK_UNARY_FALSE(__VA_ARGS__)
03040 #define REQUIRE_UNARY_FALSE(...) DOCTEST_REQUIRE_UNARY_FALSE(__VA_ARGS__)
03041
03042 // KEPT FOR BACKWARDS COMPATIBILITY
03043 #define FAST_WARN_EQ(...) DOCTEST_FAST_WARN_EQ(__VA_ARGS__)
03044 #define FAST_CHECK_EQ(...) DOCTEST_FAST_CHECK_EQ(__VA_ARGS__)
03045 #define FAST_REQUIRE_EQ(...) DOCTEST_FAST_REQUIRE_EQ(__VA_ARGS__)
03046 #define FAST_WARN_NE(...) DOCTEST_FAST_WARN_NE(__VA_ARGS__)
03047 #define FAST_CHECK_NE(...) DOCTEST_FAST_CHECK_NE(__VA_ARGS__)
03048 #define FAST_REQUIRE_NE(...) DOCTEST_FAST_REQUIRE_NE(__VA_ARGS__)
03049 #define FAST_WARN_GT(...) DOCTEST_FAST_WARN_GT(__VA_ARGS__)
03050 #define FAST_CHECK_GT(...) DOCTEST_FAST_CHECK_GT(__VA_ARGS__)
03051 #define FAST_REQUIRE_GT(...) DOCTEST_FAST_REQUIRE_GT(__VA_ARGS__)
03052 #define FAST_WARN_LT(...) DOCTEST_FAST_WARN_LT(__VA_ARGS__)
03053 #define FAST_CHECK_LT(...) DOCTEST_FAST_CHECK_LT(__VA_ARGS__)
03054 #define FAST_REQUIRE_LT(...) DOCTEST_FAST_REQUIRE_LT(__VA_ARGS__)
03055 #define FAST_WARN_GE(...) DOCTEST_FAST_WARN_GE(__VA_ARGS__)
03056 #define FAST_CHECK_GE(...) DOCTEST_FAST_CHECK_GE(__VA_ARGS__)
03057 #define FAST_REQUIRE_GE(...) DOCTEST_FAST_REQUIRE_GE(__VA_ARGS__)
03058 #define FAST_WARN_LE(...) DOCTEST_FAST_WARN_LE(__VA_ARGS__)
03059 #define FAST_CHECK_LE(...) DOCTEST_FAST_CHECK_LE(__VA_ARGS__)
03060 #define FAST_REQUIRE_LE(...) DOCTEST_FAST_REQUIRE_LE(__VA_ARGS__)
03061
03062 #define FAST_WARN_UNARY(...) DOCTEST_FAST_WARN_UNARY(__VA_ARGS__)
03063 #define FAST_CHECK_UNARY(...) DOCTEST_FAST_CHECK_UNARY(__VA_ARGS__)
03064 #define FAST_REQUIRE_UNARY(...) DOCTEST_FAST_REQUIRE_UNARY(__VA_ARGS__)
03065 #define FAST_WARN_UNARY_FALSE(...) DOCTEST_FAST_WARN_UNARY_FALSE(__VA_ARGS__)
03066 #define FAST_CHECK_UNARY_FALSE(...) DOCTEST_FAST_CHECK_UNARY_FALSE(__VA_ARGS__)
03067 #define FAST_REQUIRE_UNARY_FALSE(...) DOCTEST_FAST_REQUIRE_UNARY_FALSE(__VA_ARGS__)
03068
03069 #define TEST_CASE_TEMPLATE_INSTANTIATE(id, ...) DOCTEST_TEST_CASE_TEMPLATE_INSTANTIATE(id,
__VA_ARGS__)
03070
03071 #endif // DOCTEST_CONFIG_NO_SHORT_MACRO_NAMES
03072
03073 #ifndef DOCTEST_CONFIG_DISABLE
03074
03075 // this is here to clear the 'current test suite' for the current translation unit - at the top
03076 DOCTEST_TEST_SUITE_END();
03077
03078 #endif // DOCTEST_CONFIG_DISABLE
03079
03080 DOCTEST_CLANG_SUPPRESS_WARNING_POP
03081 DOCTEST_MSVC_SUPPRESS_WARNING_POP
03082 DOCTEST_GCC_SUPPRESS_WARNING_POP
03083
03084 DOCTEST_SUPPRESS_COMMON_WARNINGS_POP
03085
03086 #endif // DOCTEST_LIBRARY_INCLUDED
03087
03088 #ifndef DOCTEST_SINGLE_HEADER
03089 #define DOCTEST_SINGLE_HEADER
03090 #endif // DOCTEST_SINGLE_HEADER
03091
03092 #if defined(DOCTEST_CONFIG_IMPLEMENT) || !defined(DOCTEST_SINGLE_HEADER)
03093
03094 #ifndef DOCTEST_SINGLE_HEADER
03095 #include "doctest_fwd.h"
03096 #endif // DOCTEST_SINGLE_HEADER
03097
03098 DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wunused-macros")
03099
03100 #ifndef DOCTEST_LIBRARY_IMPLEMENTATION
03101 #define DOCTEST_LIBRARY_IMPLEMENTATION
03102
03103 DOCTEST_CLANG_SUPPRESS_WARNING_POP
03104
03105 DOCTEST_SUPPRESS_COMMON_WARNINGS_PUSH
03106
03107 DOCTEST_CLANG_SUPPRESS_WARNING_PUSH
03108 DOCTEST_CLANG_SUPPRESS_WARNING("-Wglobal-constructors")
03109 DOCTEST_CLANG_SUPPRESS_WARNING("-Wexit-time-destructors")
03110 DOCTEST_CLANG_SUPPRESS_WARNING("-Wsign-conversion")
03111 DOCTEST_CLANG_SUPPRESS_WARNING("-Wshorten-64-to-32")
03112 DOCTEST_CLANG_SUPPRESS_WARNING("-Wmissing-variable-declarations")
03113 DOCTEST_CLANG_SUPPRESS_WARNING("-Wswitch")
03114 DOCTEST_CLANG_SUPPRESS_WARNING("-Wswitch-enum")
03115 DOCTEST_CLANG_SUPPRESS_WARNING("-Wcovered-switch-default")
03116 DOCTEST_CLANG_SUPPRESS_WARNING("-Wmissing-noreturn")
03117 DOCTEST_CLANG_SUPPRESS_WARNING("-Wdisabled-macro-expansion")
03118 DOCTEST_CLANG_SUPPRESS_WARNING("-Wmissing-braces")
03119 DOCTEST_CLANG_SUPPRESS_WARNING("-Wmissing-field-initializers")

```

```

03120 DOCTEST_CLANG_SUPPRESS_WARNING("-Wunused-member-function")
03121 DOCTEST_CLANG_SUPPRESS_WARNING("-Wnonportable-system-include-path")
03122
03123 DOCTEST_GCC_SUPPRESS_WARNING_PUSH
03124 DOCTEST_GCC_SUPPRESS_WARNING("-Wconversion")
03125 DOCTEST_GCC_SUPPRESS_WARNING("-Wsign-conversion")
03126 DOCTEST_GCC_SUPPRESS_WARNING("-Wmissing-field-initializers")
03127 DOCTEST_GCC_SUPPRESS_WARNING("-Wmissing-braces")
03128 DOCTEST_GCC_SUPPRESS_WARNING("-Wswitch")
03129 DOCTEST_GCC_SUPPRESS_WARNING("-Wswitch-enum")
03130 DOCTEST_GCC_SUPPRESS_WARNING("-Wswitch-default")
03131 DOCTEST_GCC_SUPPRESS_WARNING("-Wunsafe-loop-optimizations")
03132 DOCTEST_GCC_SUPPRESS_WARNING("-Wold-style-cast")
03133 DOCTEST_GCC_SUPPRESS_WARNING("-Wunused-function")
03134 DOCTEST_GCC_SUPPRESS_WARNING("-Wmultiple-inheritance")
03135 DOCTEST_GCC_SUPPRESS_WARNING("-Wsuggest-attribute")
03136
03137 DOCTEST_MSVC_SUPPRESS_WARNING_PUSH
03138 DOCTEST_MSVC_SUPPRESS_WARNING(4267) // 'var' : conversion from 'x' to 'y', possible loss of data
03139 DOCTEST_MSVC_SUPPRESS_WARNING(4530) // C++ exception handler used, but unwind semantics not enabled
03140 DOCTEST_MSVC_SUPPRESS_WARNING(4577) // 'noexcept' used with no exception handling mode specified
03141 DOCTEST_MSVC_SUPPRESS_WARNING(4774) // format string expected in argument is not a string literal
03142 DOCTEST_MSVC_SUPPRESS_WARNING(4365) // conversion from 'int' to 'unsigned', signed/unsigned mismatch
03143 DOCTEST_MSVC_SUPPRESS_WARNING(5039) // pointer to potentially throwing function passed to extern C
03144 DOCTEST_MSVC_SUPPRESS_WARNING(4800) // forcing value to bool 'true' or 'false' (performance warning)
03145 DOCTEST_MSVC_SUPPRESS_WARNING(5245) // unreferenced function with internal linkage has been removed
03146
03147 DOCTEST_MAKE_STD_HEADERS_CLEAN_FROM_WARNINGS_ON_WALL_BEGIN
03148
03149 // required includes - will go only in one translation unit!
03150 #include <ctime>
03151 #include <cmath>
03152 #include <climits>
03153 // borland (Embarcadero) compiler requires math.h and not cmath -
03154 // https://github.com/doctest/doctest/pull/37
03154 #ifdef __BORLANDC__
03155 #include <math.h>
03156 #endif // __BORLANDC__
03157 #include <new>
03158 #include <cstdio>
03159 #include <cstdlib>
03160 #include <cstring>
03161 #include <limits>
03162 #include <utility>
03163 #include <fstream>
03164 #include <sstream>
03165 #ifndef DOCTEST_CONFIG_NO_INCLUDE_Iostream
03166 #include <iostream>
03167 #endif // DOCTEST_CONFIG_NO_INCLUDE_Iostream
03168 #include <algorithm>
03169 #include <iomanip>
03170 #include <vector>
03171 #ifndef DOCTEST_CONFIG_NO_MULTITHREADING
03172 #include <atomic>
03173 #include <mutex>
03174 #define DOCTEST_DECLARE_MUTEX(name) std::mutex name;
03175 #define DOCTEST_DECLARE_STATIC_MUTEX(name) static DOCTEST_DECLARE_MUTEX(name)
03176 #define DOCTEST_LOCK_MUTEX(name) std::lock_guard<std::mutex>
03177 #define DOCTEST_UNLOCK_MUTEX(name)
03178 #define DOCTEST_ANONYMOUS DOCTEST_ANON_LOCK_
03179 #else // DOCTEST_CONFIG_NO_MULTITHREADING
03180 #define DOCTEST_DECLARE_MUTEX(name)
03181 #define DOCTEST_DECLARE_STATIC_MUTEX(name)
03182 #define DOCTEST_LOCK_MUTEX(name)
03183 #define DOCTEST_UNLOCK_MUTEX(name)
03184 #endif // DOCTEST_CONFIG_NO_MULTITHREADING
03185 #include <set>
03186 #include <map>
03187 #include <unordered_set>
03188 #include <exception>
03189 #include <stdexcept>
03190 #include <csignal>
03191 #include <cfloat>
03192 #include <cctype>
03193 #include <cstdint>
03194 #include <string>
03195
03196 #ifdef DOCTEST_PLATFORM_MAC
03197 #include <sys/types.h>
03198 #include <unistd.h>
03199 #include <sys/sysctl.h>
03200 #endif // DOCTEST_PLATFORM_MAC
03201
03202 #ifdef DOCTEST_PLATFORM_WINDOWS
03203 // defines for a leaner windows.h
03204 #ifndef WIN32_LEAN_AND_MEAN
03205 #define WIN32_LEAN_AND_MEAN
03206 #endif
03207 #define DOCTEST_UNDEF_WIN32_LEAN_AND_MEAN

```



```

03205 #endif // WIN32_LEAN_AND_MEAN
03206 #ifndef NOMINMAX
03207 #define NOMINMAX
03208 #define DOCTEST_UNDEF_NOMINMAX
03209 #endif // NOMINMAX
03210
03211 // not sure what AfxWin.h is for - here I do what Catch does
03212 #ifdef __AFXDLL
03213 #include <AfxWin.h>
03214 #else
03215 #include <windows.h>
03216 #endif
03217 #include <io.h>
03218
03219 #else // DOCTEST_PLATFORM_WINDOWS
03220
03221 #include <sys/time.h>
03222 #include <unistd.h>
03223
03224 #endif // DOCTEST_PLATFORM_WINDOWS
03225
03226 // this is a fix for https://github.com/doctest/doctest/issues/348
03227 // https://mail.gnome.org/archives/xml/2012-January/msg00000.html
03228 #if !defined(HAVE_UNISTD_H) && !defined(STDOUT_FILENO)
03229 #define STDOUT_FILENO fileno(stdout)
03230 #endif // HAVE_UNISTD_H
03231
03232 DOCTEST_MAKE_STD_HEADERS_CLEAN_FROM_WARNINGS_ON_WALL_END
03233
03234 // counts the number of elements in a C array
03235 #define DOCTEST_COUNTOF(x) (sizeof(x) / sizeof(x[0]))
03236
03237 #ifdef DOCTEST_CONFIG_DISABLE
03238 #define DOCTEST_BRANCH_ON_DISABLED(if_disabled, if_not_disabled) if_disabled
03239 #else // DOCTEST_CONFIG_DISABLE
03240 #define DOCTEST_BRANCH_ON_DISABLED(if_disabled, if_not_disabled) if_not_disabled
03241 #endif // DOCTEST_CONFIG_DISABLE
03242
03243 #ifndef DOCTEST_CONFIG_OPTIONS_PREFIX
03244 #define DOCTEST_CONFIG_OPTIONS_PREFIX "dt-"
03245 #endif
03246
03247 #ifndef DOCTEST_CONFIG_OPTIONS_FILE_PREFIX_SEPARATOR
03248 #define DOCTEST_CONFIG_OPTIONS_FILE_PREFIX_SEPARATOR ':'
03249 #endif
03250
03251 #ifndef DOCTEST_THREAD_LOCAL
03252 #if defined(DOCTEST_CONFIG_NO_MULTITHREADING) || DOCTEST_MSVC && (DOCTEST_MSVC < DOCTEST_COMPILER(19,
0, 0))
03253 #define DOCTEST_THREAD_LOCAL
03254 #else // DOCTEST_MSVC
03255 #define DOCTEST_THREAD_LOCAL thread_local
03256 #endif // DOCTEST_MSVC
03257 #endif // DOCTEST_THREAD_LOCAL
03258
03259 #ifndef DOCTEST_MULTI_LANE_ATOMICS_THREAD_LANES
03260 #define DOCTEST_MULTI_LANE_ATOMICS_THREAD_LANES 32
03261 #endif
03262
03263 #ifndef DOCTEST_MULTI_LANE_ATOMICS_CACHE_LINE_SIZE
03264 #define DOCTEST_MULTI_LANE_ATOMICS_CACHE_LINE_SIZE 64
03265 #endif
03266
03267 #ifdef DOCTEST_CONFIG_NO_UNPREFIXED_OPTIONS
03268 #define DOCTEST_OPTIONS_PREFIX_DISPLAY DOCTEST_CONFIG_OPTIONS_PREFIX
03269 #else
03270 #define DOCTEST_OPTIONS_PREFIX_DISPLAY ""
03271 #endif
03272
03273 #if defined(WINAPI_FAMILY) && (WINAPI_FAMILY == WINAPI_FAMILY_APP)
03274 #define DOCTEST_CONFIG_NO_MULTI_LANE_ATOMICS
03275 #endif
03276
03277 #ifndef DOCTEST_CDECL
03278 #define DOCTEST_CDECL __cdecl
03279 #endif
03280
03281 namespace doctest {
03282
03283 bool is_running_in_test = false;
03284
03285 namespace {
03286     using namespace detail;
03287
03288     template <typename Ex>
03289     DOCTEST_NORETURN void throw_exception(Ex const& e) {
03290 #ifndef DOCTEST_CONFIG_NO_EXCEPTIONS

```

```

03291         throw e;
03292 #else // DOCTEST_CONFIG_NO_EXCEPTIONS
03293 #ifdef DOCTEST_CONFIG_HANDLE_EXCEPTION
03294     DOCTEST_CONFIG_HANDLE_EXCEPTION(e);
03295 #else // DOCTEST_CONFIG_HANDLE_EXCEPTION
03296 #ifndef DOCTEST_CONFIG_NO_INCLUDE_Iostream
03297     std::cerr << "doctest will terminate because it needed to throw an exception.\n"
03298               << "The message was: " << e.what() << '\n';
03299 #endif // DOCTEST_CONFIG_NO_INCLUDE_Iostream
03300 #endif // DOCTEST_CONFIG_HANDLE_EXCEPTION
03301     std::terminate();
03302 #endif // DOCTEST_CONFIG_NO_EXCEPTIONS
03303 }
03304
03305 #ifndef DOCTEST_INTERNAL_ERROR
03306 #define DOCTEST_INTERNAL_ERROR(msg)
03307     throw_exception(std::logic_error(
03308         __FILE__ ":" DOCTEST_TOSTR(__LINE__) ": Internal doctest error: " msg))
03309 #endif // DOCTEST_INTERNAL_ERROR
03310
03311 // case insensitive strcmp
03312 int stricmp(const char* a, const char* b) {
03313     for(;; a++, b++) {
03314         const int d = tolower(*a) - tolower(*b);
03315         if(d != 0 || !*a)
03316             return d;
03317     }
03318 }
03319
03320 struct Endianness
03321 {
03322     enum Arch
03323     {
03324         Big,
03325         Little
03326     };
03327
03328     static Arch which() {
03329         int x = 1;
03330         // casting any data pointer to char* is allowed
03331         auto ptr = reinterpret_cast<char*>(&x);
03332         if(*ptr)
03333             return Little;
03334         return Big;
03335     }
03336 };
03337 } // namespace
03338
03339 namespace detail {
03340     DOCTEST_THREAD_LOCAL class
03341     {
03342     public:
03343         std::vector<std::streampos> stack;
03344         std::stringstream ss;
03345
03346         std::ostream* push() {
03347             stack.push_back(ss.tellp());
03348             return &ss;
03349         }
03350
03351         String pop() {
03352             if (stack.empty())
03353                 DOCTEST_INTERNAL_ERROR("TLSS was empty when trying to pop!");
03354
03355             std::streampos pos = stack.back();
03356             stack.pop_back();
03357             unsigned sz = static_cast<unsigned>(ss.tellp() - pos);
03358             ss.rdbuf()->pubseekpos(pos, std::ios::in | std::ios::out);
03359             return String(ss, sz);
03360         }
03361     } g_oss;
03362
03363     std::ostream* tlssPush() {
03364         return g_oss.push();
03365     }
03366
03367     String tlssPop() {
03368         return g_oss.pop();
03369     }
03370
03371 #ifndef DOCTEST_CONFIG_DISABLE
03372 namespace timer_large_integer
03373 {
03374     #if defined(DOCTEST_PLATFORM_WINDOWS)
03375     using type = ULONGLONG;
03376     #endif
03377 }

```

```

03378 #else // DOCTEST_PLATFORM_WINDOWS
03379     using type = std::uint64_t;
03380 #endif // DOCTEST_PLATFORM_WINDOWS
03381 }
03382
03383 using ticks_t = timer_large_integer::type;
03384
03385 #ifndef DOCTEST_CONFIG_GETCURRENTTICKS
03386     ticks_t getCurrentTicks() { return DOCTEST_CONFIG_GETCURRENTTICKS(); }
03387 #elif defined(DOCTEST_PLATFORM_WINDOWS)
03388     ticks_t getCurrentTicks() {
03389         static LARGE_INTEGER hz = { {0} }, hzo = { {0} };
03390         if(!hz.QuadPart) {
03391             QueryPerformanceFrequency(&hz);
03392             QueryPerformanceCounter(&hzo);
03393         }
03394         LARGE_INTEGER t;
03395         QueryPerformanceCounter(&t);
03396         return ((t.QuadPart - hzo.QuadPart) * LONGLONG(1000000)) / hz.QuadPart;
03397     }
03398 #else // DOCTEST_PLATFORM_WINDOWS
03399     ticks_t getCurrentTicks() {
03400         timeval t;
03401         gettimeofday(&t, nullptr);
03402         return static_cast<ticks_t>(t.tv_sec) * 1000000 + static_cast<ticks_t>(t.tv_usec);
03403     }
03404 #endif // DOCTEST_PLATFORM_WINDOWS
03405
03406 struct Timer
03407 {
03408     void start() { m_ticks = getCurrentTicks(); }
03409     unsigned int getElapsedMicroseconds() const {
03410         return static_cast<unsigned int>(getCurrentTicks() - m_ticks);
03411     }
03412     //unsigned int getElapsedMilliseconds() const {
03413     //    return static_cast<unsigned int>(getElapsedMicroseconds() / 1000);
03414     //}
03415     double getElapsedSeconds() const { return static_cast<double>(getCurrentTicks() - m_ticks) /
1000000.0; }
03416
03417 private:
03418     ticks_t m_ticks = 0;
03419 };
03420
03421 #ifndef DOCTEST_CONFIG_NO_MULTITHREADING
03422     template <typename T>
03423     using Atomic = T;
03424 #else // DOCTEST_CONFIG_NO_MULTITHREADING
03425     template <typename T>
03426     using Atomic = std::atomic<T>;
03427 #endif // DOCTEST_CONFIG_NO_MULTITHREADING
03428
03429 #if defined(DOCTEST_CONFIG_NO_MULTI_LANE_ATOMICS) || defined(DOCTEST_CONFIG_NO_MULTITHREADING)
03430     template <typename T>
03431     using MultiLaneAtomic = Atomic<T>;
03432 #else // DOCTEST_CONFIG_NO_MULTI_LANE_ATOMICS
03433     // Provides a multilane implementation of an atomic variable that supports add, sub, load,
03434     // store. Instead of using a single atomic variable, this splits up into multiple ones,
03435     // each sitting on a separate cache line. The goal is to provide a speedup when most
03436     // operations are modifying. It achieves this with two properties:
03437     //
03438     // * Multiple atomics are used, so chance of congestion from the same atomic is reduced.
03439     // * Each atomic sits on a separate cache line, so false sharing is reduced.
03440     //
03441     // The disadvantage is that there is a small overhead due to the use of TLS, and load/store
03442     // is slower because all atomics have to be accessed.
03443     template <typename T>
03444     class MultiLaneAtomic
03445     {
03446     public:
03447         struct CacheLineAlignedAtomic
03448         {
03449             Atomic<T> atomic{};
03450             char padding[DOCTEST_MULTI_LANE_ATOMICS_CACHE_LINE_SIZE - sizeof(Atomic<T>)];
03451         };
03452         CacheLineAlignedAtomic m_atomics[DOCTEST_MULTI_LANE_ATOMICS_THREAD_LANES];
03453
03454         static_assert(sizeof(CacheLineAlignedAtomic) == DOCTEST_MULTI_LANE_ATOMICS_CACHE_LINE_SIZE,
03455             "guarantee one atomic takes exactly one cache line");
03456
03457         T operator++() DOCTEST_NOEXCEPT { return fetch_add(1) + 1; }
03458
03459         T operator++(int) DOCTEST_NOEXCEPT { return fetch_add(1); }
03460
03461         T fetch_add(T arg, std::memory_order order = std::memory_order_seq_cst) DOCTEST_NOEXCEPT {
03462             return myAtomic().fetch_add(arg, order);
03463         }
03464     };

```

```

03464
03465     T fetch_sub(T arg, std::memory_order order = std::memory_order_seq_cst) DOCTEST_NOEXCEPT {
03466         return myAtomic().fetch_sub(arg, order);
03467     }
03468
03469     operator T() const DOCTEST_NOEXCEPT { return load(); }
03470
03471     T load(std::memory_order order = std::memory_order_seq_cst) const DOCTEST_NOEXCEPT {
03472         auto result = T();
03473         for(auto const& c : m_atomics) {
03474             result += c.atomic.load(order);
03475         }
03476         return result;
03477     }
03478
03479     T operator=(T desired) DOCTEST_NOEXCEPT { // lgtm [cpp/assignment-does-not-return-this]
03480         store(desired);
03481         return desired;
03482     }
03483
03484     void store(T desired, std::memory_order order = std::memory_order_seq_cst) DOCTEST_NOEXCEPT {
03485         // first value becomes desired", all others become 0.
03486         for(auto& c : m_atomics) {
03487             c.atomic.store(desired, order);
03488             desired = {};
03489         }
03490     }
03491
03492 private:
03493     // Each thread has a different atomic that it operates on. If more than NumLanes threads
03494     // use this, some will use the same atomic. So performance will degrade a bit, but still
03495     // everything will work.
03496     //
03497     // The logic here is a bit tricky. The call should be as fast as possible, so that there
03498     // is minimal to no overhead in determining the correct atomic for the current thread.
03499     //
03500     // 1. A global static counter laneCounter counts continuously up.
03501     // 2. Each successive thread will use modulo operation of that counter so it gets an atomic
03502     //    assigned in a round-robin fashion.
03503     // 3. This tlsLaneIdx is stored in the thread local data, so it is directly available with
03504     //    little overhead.
03505     Atomic<T>& myAtomic() DOCTEST_NOEXCEPT {
03506         static Atomic<size_t> laneCounter;
03507         DOCTEST_THREAD_LOCAL size_t tlsLaneIdx =
03508             laneCounter++ % DOCTEST_MULTI_LANE_ATOMICS_THREAD_LANES;
03509
03510         return m_atomics[tlsLaneIdx].atomic;
03511     }
03512 };
03513 #endif // DOCTEST_CONFIG_NO_MULTI_LANE_ATOMICS
03514
03515 // this holds both parameters from the command line and runtime data for tests
03516 struct ContextState : ContextOptions, TestRunStats, CurrentTestCaseStats
03517 {
03518     MultiLaneAtomic<int> numAssertsCurrentTest_atomic;
03519     MultiLaneAtomic<int> numAssertsFailedCurrentTest_atomic;
03520
03521     std::vector<std::vector<String>> filters = decltype(filters)(9); // 9 different filters
03522
03523     std::vector<IReporter*> reporters_currently_used;
03524
03525     assert_handler ah = nullptr;
03526
03527     Timer timer;
03528
03529     std::vector<String> stringifiedContexts; // logging from INFO() due to an exception
03530
03531     // stuff for subcases
03532     bool reachedLeaf;
03533     std::vector<SubcaseSignature> subcaseStack;
03534     std::vector<SubcaseSignature> nextSubcaseStack;
03535     std::unordered_set<unsigned long long> fullyTraversedSubcases;
03536     size_t currentSubcaseDepth;
03537     Atomic<bool> shouldLogCurrentException;
03538
03539     void resetRunData() {
03540         numTestCases = 0;
03541         numTestCasesPassingFilters = 0;
03542         numTestSuitesPassingFilters = 0;
03543         numTestCasesFailed = 0;
03544         numAsserts = 0;
03545         numAssertsFailed = 0;
03546         numAssertsCurrentTest = 0;
03547         numAssertsFailedCurrentTest = 0;
03548     }
03549
03550     void finalizeTestCaseData() {

```

```

03551         seconds = timer.getElapsedSeconds();
03552
03553         // update the non-atomic counters
03554         numAsserts += numAssertsCurrentTest_atomic;
03555         numAssertsFailed += numAssertsFailedCurrentTest_atomic;
03556         numAssertsCurrentTest = numAssertsCurrentTest_atomic;
03557         numAssertsFailedCurrentTest = numAssertsFailedCurrentTest_atomic;
03558
03559         if(numAssertsFailedCurrentTest)
03560             failure_flags |= TestCaseFailureReason::AssertFailure;
03561
03562         if(Approx(currentTest->m_timeout).epsilon(DBL_EPSILON) != 0 &&
03563            Approx(seconds).epsilon(DBL_EPSILON) > currentTest->m_timeout)
03564             failure_flags |= TestCaseFailureReason::Timeout;
03565
03566         if(currentTest->m_should_fail) {
03567             if(failure_flags) {
03568                 failure_flags |= TestCaseFailureReason::ShouldHaveFailedAndDid;
03569             } else {
03570                 failure_flags |= TestCaseFailureReason::ShouldHaveFailedButDidnt;
03571             }
03572         } else if(failure_flags && currentTest->m_may_fail) {
03573             failure_flags |= TestCaseFailureReason::CouldHaveFailedAndDid;
03574         } else if(currentTest->m_expected_failures > 0) {
03575             if(numAssertsFailedCurrentTest == currentTest->m_expected_failures) {
03576                 failure_flags |= TestCaseFailureReason::FailedExactlyNumTimes;
03577             } else {
03578                 failure_flags |= TestCaseFailureReason::DidntFailExactlyNumTimes;
03579             }
03580         }
03581
03582         bool ok_to_fail = (TestCaseFailureReason::ShouldHaveFailedAndDid & failure_flags) ||
03583             (TestCaseFailureReason::CouldHaveFailedAndDid & failure_flags) ||
03584             (TestCaseFailureReason::FailedExactlyNumTimes & failure_flags);
03585
03586         // if any subcase has failed - the whole test case has failed
03587         testCaseSuccess = !(failure_flags && !ok_to_fail);
03588         if(!testCaseSuccess)
03589             numTestCasesFailed++;
03590     }
03591 };
03592
03593 ContextState* g_cs = nullptr;
03594
03595 // used to avoid locks for the debug output
03596 // TODO: figure out if this is indeed necessary/correct - seems like either there still
03597 // could be a race or that there wouldn't be a race even if using the context directly
03598 DOCTEST_THREAD_LOCAL bool g_no_colors;
03599
03600 #endif // DOCTEST_CONFIG_DISABLE
03601 } // namespace detail
03602
03603 char* String::allocate(size_type sz) {
03604     if (sz <= last) {
03605         buf[sz] = '\0';
03606         setLast(last - sz);
03607         return buf;
03608     } else {
03609         setOnHeap();
03610         data.size = sz;
03611         data.capacity = data.size + 1;
03612         data.ptr = new char[data.capacity];
03613         data.ptr[sz] = '\0';
03614         return data.ptr;
03615     }
03616 }
03617
03618 void String::setOnHeap() noexcept { *reinterpret_cast<unsigned char*>(&buf[last]) = 128; }
03619 void String::setLast(size_type in) noexcept { buf[last] = char(in); }
03620 void String::setSize(size_type sz) noexcept {
03621     if (isOnStack()) { buf[sz] = '\0'; setLast(last - sz); }
03622     else { data.ptr[sz] = '\0'; data.size = sz; }
03623 }
03624
03625 void String::copy(const String& other) {
03626     if(other.isOnStack()) {
03627         memcpy(buf, other.buf, len);
03628     } else {
03629         memcpy(allocate(other.data.size), other.data.ptr, other.data.size);
03630     }
03631 }
03632
03633 String::String() noexcept {
03634     buf[0] = '\0';
03635     setLast();
03636 }
03637

```

```

03638 String::~String() {
03639     if(!isOnStack())
03640         delete[] data.ptr;
03641 } // NOLINT(clang-analyzer-cplusplus.NewDeleteLeaks)
03642
03643 String::String(const char* in)
03644     : String(in, strlen(in)) {}
03645
03646 String::String(const char* in, size_type in_size) {
03647     memcpy(allocate(in_size), in, in_size);
03648 }
03649
03650 String::String(std::istream& in, size_type in_size) {
03651     in.read(allocate(in_size), in_size);
03652 }
03653
03654 String::String(const String& other) { copy(other); }
03655
03656 String& String::operator=(const String& other) {
03657     if(this != &other) {
03658         if(!isOnStack())
03659             delete[] data.ptr;
03660
03661         copy(other);
03662     }
03663
03664     return *this;
03665 }
03666
03667 String& String::operator+=(const String& other) {
03668     const size_type my_old_size = size();
03669     const size_type other_size = other.size();
03670     const size_type total_size = my_old_size + other_size;
03671     if(isOnStack()) {
03672         if(total_size < len) {
03673             // append to the current stack space
03674             memcpy(buf + my_old_size, other.c_str(), other_size + 1);
03675             // NOLINTNEXTLINE(clang-analyzer-cplusplus.NewDeleteLeaks)
03676             setLast(last - total_size);
03677         } else {
03678             // alloc new chunk
03679             char* temp = new char[total_size + 1];
03680             // copy current data to new location before writing in the union
03681             memcpy(temp, buf, my_old_size); // skip the +1 ('\0') for speed
03682             // update data in union
03683             setOnHeap();
03684             data.size = total_size;
03685             data.capacity = data.size + 1;
03686             data.ptr = temp;
03687             // transfer the rest of the data
03688             memcpy(data.ptr + my_old_size, other.c_str(), other_size + 1);
03689         }
03690     } else {
03691         if(data.capacity > total_size) {
03692             // append to the current heap block
03693             data.size = total_size;
03694             memcpy(data.ptr + my_old_size, other.c_str(), other_size + 1);
03695         } else {
03696             // resize
03697             data.capacity *= 2;
03698             if(data.capacity <= total_size)
03699                 data.capacity = total_size + 1;
03700             // alloc new chunk
03701             char* temp = new char[data.capacity];
03702             // copy current data to new location before releasing it
03703             memcpy(temp, data.ptr, my_old_size); // skip the +1 ('\0') for speed
03704             // release old chunk
03705             delete[] data.ptr;
03706             // update the rest of the union members
03707             data.size = total_size;
03708             data.ptr = temp;
03709             // transfer the rest of the data
03710             memcpy(data.ptr + my_old_size, other.c_str(), other_size + 1);
03711         }
03712     }
03713
03714     return *this;
03715 }
03716
03717 String::String(String&& other) noexcept {
03718     memcpy(buf, other.buf, len);
03719     other.buf[0] = '\0';
03720     other.setLast();
03721 }
03722
03723 String& String::operator=(String&& other) noexcept {
03724     if(this != &other) {

```

```

03725         if(!isOnStack())
03726             delete[] data.ptr;
03727         memcpy(buf, other.buf, len);
03728         other.buf[0] = '\0';
03729         other.setLast();
03730     }
03731     return *this;
03732 }
03733
03734 char String::operator[](size_type i) const {
03735     return const_cast<String*>(this)->operator[](i);
03736 }
03737
03738 char& String::operator[](size_type i) {
03739     if(isOnStack())
03740         return reinterpret_cast<char*>(buf)[i];
03741     return data.ptr[i];
03742 }
03743
03744 DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH("-Wmaybe-uninitialized")
03745 String::size_type String::size() const {
03746     if(isOnStack())
03747         return last - (size_type(buf[last]) & 31); // using "last" would work only if "len" is 32
03748     return data.size;
03749 }
03750 DOCTEST_GCC_SUPPRESS_WARNING_POP
03751
03752 String::size_type String::capacity() const {
03753     if(isOnStack())
03754         return len;
03755     return data.capacity;
03756 }
03757
03758 String String::substr(size_type pos, size_type cnt) && {
03759     cnt = std::min(cnt, size() - pos);
03760     char* cptr = c_str();
03761     memmove(cptr, cptr + pos, cnt);
03762     setSize(cnt);
03763     return std::move(*this);
03764 }
03765
03766 String String::substr(size_type pos, size_type cnt) const & {
03767     cnt = std::min(cnt, size() - pos);
03768     return String{ c_str() + pos, cnt };
03769 }
03770
03771 String::size_type String::find(char ch, size_type pos) const {
03772     const char* begin = c_str();
03773     const char* end = begin + size();
03774     const char* it = begin + pos;
03775     for (; it < end && *it != ch; it++);
03776     if (it < end) { return static_cast<size_type>(it - begin); }
03777     else { return npos; }
03778 }
03779
03780 String::size_type String::rfind(char ch, size_type pos) const {
03781     const char* begin = c_str();
03782     const char* it = begin + std::min(pos, size() - 1);
03783     for (; it >= begin && *it != ch; it--);
03784     if (it >= begin) { return static_cast<size_type>(it - begin); }
03785     else { return npos; }
03786 }
03787
03788 int String::compare(const char* other, bool no_case) const {
03789     if(no_case)
03790         return doctest::stricmp(c_str(), other);
03791     return std::strcmp(c_str(), other);
03792 }
03793
03794 int String::compare(const String& other, bool no_case) const {
03795     return compare(other.c_str(), no_case);
03796 }
03797
03798 String operator+(const String& lhs, const String& rhs) { return String(lhs) += rhs; }
03799
03800 bool operator==(const String& lhs, const String& rhs) { return lhs.compare(rhs) == 0; }
03801 bool operator!=(const String& lhs, const String& rhs) { return lhs.compare(rhs) != 0; }
03802 bool operator<(const String& lhs, const String& rhs) { return lhs.compare(rhs) < 0; }
03803 bool operator>(const String& lhs, const String& rhs) { return lhs.compare(rhs) > 0; }
03804 bool operator<=(const String& lhs, const String& rhs) { return (lhs != rhs) ? lhs.compare(rhs) < 0 :
    true; }
03805 bool operator>=(const String& lhs, const String& rhs) { return (lhs != rhs) ? lhs.compare(rhs) > 0 :
    true; }
03806
03807 std::ostream& operator<<(std::ostream& s, const String& in) { return s << in.c_str(); }
03808
03809 Contains::Contains(const String& str) : string(str) { }

```

```

03810
03811 bool Contains::checkWith(const String& other) const {
03812     return strstr(other.c_str(), string.c_str()) != nullptr;
03813 }
03814
03815 String toString(const Contains& in) {
03816     return "Contains( " + in.string + " )";
03817 }
03818
03819 bool operator==(const String& lhs, const Contains& rhs) { return rhs.checkWith(lhs); }
03820 bool operator==(const Contains& lhs, const String& rhs) { return lhs.checkWith(rhs); }
03821 bool operator!=(const String& lhs, const Contains& rhs) { return !rhs.checkWith(lhs); }
03822 bool operator!=(const Contains& lhs, const String& rhs) { return !lhs.checkWith(rhs); }
03823
03824 namespace {
03825     void color_to_stream(std::ostream&, Color::Enum) DOCTEST_BRANCH_ON_DISABLED({}, ; )
03826 } // namespace
03827
03828 namespace Color {
03829     std::ostream& operator<<(std::ostream& s, Color::Enum code) {
03830         color_to_stream(s, code);
03831         return s;
03832     }
03833 } // namespace Color
03834
03835 // clang-format off
03836 const char* assertString(assertType::Enum at) {
03837     DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4061) // enum 'x' in switch of enum 'y' is not explicitly
03838     handled
03839     #define DOCTEST_GENERATE_ASSERT_TYPE_CASE(assert_type) case assertType::DT_ ## assert_type: return
03840     #define DOCTEST_GENERATE_ASSERT_TYPE_CASES(assert_type) \
03841         DOCTEST_GENERATE_ASSERT_TYPE_CASE(WARN_ ## assert_type); \
03842         DOCTEST_GENERATE_ASSERT_TYPE_CASE(CHECK_ ## assert_type); \
03843         DOCTEST_GENERATE_ASSERT_TYPE_CASE(REQUIRE_ ## assert_type)
03844     switch(at) {
03845         DOCTEST_GENERATE_ASSERT_TYPE_CASE(WARN);
03846         DOCTEST_GENERATE_ASSERT_TYPE_CASE(CHECK);
03847         DOCTEST_GENERATE_ASSERT_TYPE_CASE(REQUIRE);
03848         DOCTEST_GENERATE_ASSERT_TYPE_CASES(FALSE);
03849         DOCTEST_GENERATE_ASSERT_TYPE_CASES(THROWS);
03850         DOCTEST_GENERATE_ASSERT_TYPE_CASES(THROWS_AS);
03851         DOCTEST_GENERATE_ASSERT_TYPE_CASES(THROWS_WITH);
03852         DOCTEST_GENERATE_ASSERT_TYPE_CASES(THROWS_WITH_AS);
03853         DOCTEST_GENERATE_ASSERT_TYPE_CASES(NOTHROW);
03854         DOCTEST_GENERATE_ASSERT_TYPE_CASES(EQ);
03855         DOCTEST_GENERATE_ASSERT_TYPE_CASES(NE);
03856         DOCTEST_GENERATE_ASSERT_TYPE_CASES(GT);
03857         DOCTEST_GENERATE_ASSERT_TYPE_CASES(LT);
03858         DOCTEST_GENERATE_ASSERT_TYPE_CASES(GE);
03859         DOCTEST_GENERATE_ASSERT_TYPE_CASES(LE);
03860         DOCTEST_GENERATE_ASSERT_TYPE_CASES(UNARY);
03861         DOCTEST_GENERATE_ASSERT_TYPE_CASES(UNARY_FALSE);
03862         default: DOCTEST_INTERNAL_ERROR("Tried stringifying invalid assert type!");
03863     }
03864     DOCTEST_MSVC_SUPPRESS_WARNING_POP
03865 }
03866 // clang-format on
03867
03868 const char* failureString(assertType::Enum at) {
03869     if(at & assertType::is_warn)
03870         return "WARNING";
03871     if(at & assertType::is_check)
03872         return "ERROR";
03873     if(at & assertType::is_require)
03874         return "FATAL ERROR";
03875     return "";
03876 }
03877
03878 DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wnull-dereference")
03879 DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH("-Wnull-dereference")
03880 // depending on the current options this will remove the path of filenames
03881 const char* skipPathFromFilename(const char* file) {
03882     #ifndef DOCTEST_CONFIG_DISABLE
03883     if(getContextOptions()->no_path_in_filenames) {
03884         auto back = std::strrchr(file, '\\');
03885         auto forward = std::strrchr(file, '/');
03886         if(back || forward) {

```



```

03895         if(back > forward)
03896             forward = back;
03897         return forward + 1;
03898     }
03899 } else {
03900     const auto prefixes = getContextOptions()->strip_file_prefixes;
03901     const char separator = DOCTEST_CONFIG_OPTIONS_FILE_PREFIX_SEPARATOR;
03902     String::size_type longest_match = 0U;
03903     for(String::size_type pos = 0U; pos < prefixes.size(); ++pos)
03904     {
03905         const auto prefix_start = pos;
03906         pos = std::min(prefixes.find(separator, prefix_start), prefixes.size());
03907
03908         const auto prefix_size = pos - prefix_start;
03909         if(prefix_size > longest_match)
03910         {
03911             // TODO under DOCTEST_MSVC: does the comparison need strnicmp() to work with drive
03912             letter capitalization?
03913             if(0 == std::strncmp(prefixes.c_str() + prefix_start, file, prefix_size))
03914             {
03915                 longest_match = prefix_size;
03916             }
03917         }
03918         return &file[longest_match];
03919     }
03920 #endif // DOCTEST_CONFIG_DISABLE
03921     return file;
03922 }
03923 DOCTEST_CLANG_SUPPRESS_WARNING_POP
03924 DOCTEST_GCC_SUPPRESS_WARNING_POP
03925
03926 bool SubcaseSignature::operator==(const SubcaseSignature& other) const {
03927     return m_line == other.m_line
03928         && std::strcmp(m_file, other.m_file) == 0
03929         && m_name == other.m_name;
03930 }
03931
03932 bool SubcaseSignature::operator<(const SubcaseSignature& other) const {
03933     if(m_line != other.m_line)
03934         return m_line < other.m_line;
03935     if(std::strcmp(m_file, other.m_file) != 0)
03936         return std::strcmp(m_file, other.m_file) < 0;
03937     return m_name.compare(other.m_name) < 0;
03938 }
03939
03940 DOCTEST_DEFINE_INTERFACE(IcontextScope)
03941
03942 namespace detail {
03943     void filldata<const void*>::fill(std::ostream* stream, const void* in) {
03944         if(in) { *stream << in; }
03945         else { *stream << "nullptr"; }
03946     }
03947
03948     template <typename T>
03949     String toStreamLit(T t) {
03950         std::ostream* os = tlssPush();
03951         os->operator<<(t);
03952         return tlssPop();
03953     }
03954 }
03955
03956 #ifdef DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
03957 String toString(const char* in) { return String("\0") + (in ? in : "{null string}") + "\0"; }
03958 #endif // DOCTEST_CONFIG_TREAT_CHAR_STAR_AS_STRING
03959
03960 #if DOCTEST_MSVC >= DOCTEST_COMPILER(19, 20, 0)
03961 // see this issue on why this is needed: https://github.com/doctest/doctest/issues/183
03962 String toString(const std::string& in) { return in.c_str(); }
03963 #endif // VS 2019
03964
03965 String toString(String in) { return in; }
03966
03967 String toString(std::nullptr_t) { return "nullptr"; }
03968
03969 String toString(bool in) { return in ? "true" : "false"; }
03970
03971 String toString(float in) { return toStreamLit(in); }
03972 String toString(double in) { return toStreamLit(in); }
03973 String toString(double long in) { return toStreamLit(in); }
03974
03975 String toString(char in) { return toStreamLit(static_cast<signed>(in)); }
03976 String toString(char signed in) { return toStreamLit(static_cast<signed>(in)); }
03977 String toString(char unsigned in) { return toStreamLit(static_cast<unsigned>(in)); }
03978 String toString(short in) { return toStreamLit(in); }
03979 String toString(short unsigned in) { return toStreamLit(in); }
03980 String toString(signed in) { return toStreamLit(in); }

```

```

03981 String toString(unsigned in) { return toStringLit(in); }
03982 String toString(long in) { return toStringLit(in); }
03983 String toString(long unsigned in) { return toStringLit(in); }
03984 String toString(long long in) { return toStringLit(in); }
03985 String toString(long long unsigned in) { return toStringLit(in); }
03986
03987 Approx::Approx(double value)
03988     : m_epsilon(static_cast<double>(std::numeric_limits<float>::epsilon()) * 100)
03989     , m_scale(1.0)
03990     , m_value(value) {}
03991
03992 Approx Approx::operator()(double value) const {
03993     Approx approx(value);
03994     approx.epsilon(m_epsilon);
03995     approx.scale(m_scale);
03996     return approx;
03997 }
03998
03999 Approx& Approx::epsilon(double newEpsilon) {
04000     m_epsilon = newEpsilon;
04001     return *this;
04002 }
04003 Approx& Approx::scale(double newScale) {
04004     m_scale = newScale;
04005     return *this;
04006 }
04007
04008 bool operator==(double lhs, const Approx& rhs) {
04009     // Thanks to Richard Harris for his help refining this formula
04010     return std::fabs(lhs - rhs.m_value) <
04011         rhs.m_epsilon * (rhs.m_scale + std::max<double>(std::fabs(lhs), std::fabs(rhs.m_value)));
04012 }
04013 bool operator==(const Approx& lhs, double rhs) { return operator==(rhs, lhs); }
04014 bool operator!=(double lhs, const Approx& rhs) { return !operator==(lhs, rhs); }
04015 bool operator!=(const Approx& lhs, double rhs) { return !operator==(rhs, lhs); }
04016 bool operator<=(double lhs, const Approx& rhs) { return lhs < rhs.m_value || lhs == rhs; }
04017 bool operator<=(const Approx& lhs, double rhs) { return lhs.m_value < rhs || lhs == rhs; }
04018 bool operator>=(double lhs, const Approx& rhs) { return lhs > rhs.m_value || lhs == rhs; }
04019 bool operator>=(const Approx& lhs, double rhs) { return lhs.m_value > rhs || lhs == rhs; }
04020 bool operator<(double lhs, const Approx& rhs) { return lhs < rhs.m_value && lhs != rhs; }
04021 bool operator<(const Approx& lhs, double rhs) { return lhs.m_value < rhs && lhs != rhs; }
04022 bool operator>(double lhs, const Approx& rhs) { return lhs > rhs.m_value && lhs != rhs; }
04023 bool operator>(const Approx& lhs, double rhs) { return lhs.m_value > rhs && lhs != rhs; }
04024
04025 String toString(const Approx& in) {
04026     return "Approx( " + doctest::toString(in.m_value) + " )";
04027 }
04028 const ContextOptions* getContextOptions() { return DOCTEST_BRANCH_ON_DISABLED(nullptr, g_cs); }
04029
04030 DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4738)
04031 template <typename F>
04032 IsNaN<F>::operator bool() const {
04033     return std::isnan(value) ^ flipped;
04034 }
04035 DOCTEST_MSVC_SUPPRESS_WARNING_POP
04036 template struct DOCTEST_INTERFACE_DEF IsNaN<float>;
04037 template struct DOCTEST_INTERFACE_DEF IsNaN<double>;
04038 template struct DOCTEST_INTERFACE_DEF IsNaN<long double>;
04039 template <typename F>
04040 String toString(IsNaN<F> in) { return String(in.flipped ? "!" : "") + "IsNaN( " +
doctest::toString(in.value) + " )"; }
04041 String toString(IsNaN<float> in) { return toString<float>(in); }
04042 String toString(IsNaN<double> in) { return toString<double>(in); }
04043 String toString(IsNaN<double long> in) { return toString<double long>(in); }
04044
04045 } // namespace doctest
04046
04047 #ifdef DOCTEST_CONFIG_DISABLE
04048 namespace doctest {
04049 Context::Context(int, const char* const*) {}
04050 Context::~Context() = default;
04051 void Context::applyCommandLine(int, const char* const*) {}
04052 void Context::addFilter(const char*, const char*) {}
04053 void Context::clearFilters() {}
04054 void Context::setOption(const char*, bool) {}
04055 void Context::setOption(const char*, int) {}
04056 void Context::setOption(const char*, const char*) {}
04057 bool Context::shouldExit() { return false; }
04058 void Context::setAsDefaultForAssertsOutOfTestCases() {}
04059 void Context::setAssertHandler(detail::assert_handler) {}
04060 void Context::setCout(std::ostream*) {}
04061 int Context::run() { return 0; }
04062
04063 int IReporter::get_num_active_contexts() { return 0; }
04064 const IContextScope* const* IReporter::get_active_contexts() { return nullptr; }
04065 int IReporter::get_num_stringified_contexts() { return 0; }
04066 const String* IReporter::get_stringified_contexts() { return nullptr; }

```

```

04067
04068 int registerReporter(const char*, int, IReporter*) { return 0; }
04069
04070 } // namespace doctest
04071 #else // DOCTEST_CONFIG_DISABLE
04072
04073 #if !defined(DOCTEST_CONFIG_COLORS_NONE)
04074 #if !defined(DOCTEST_CONFIG_COLORS_WINDOWS) && !defined(DOCTEST_CONFIG_COLORS_ANSI)
04075 #ifdef DOCTEST_PLATFORM_WINDOWS
04076 #define DOCTEST_CONFIG_COLORS_WINDOWS
04077 #else // linux
04078 #define DOCTEST_CONFIG_COLORS_ANSI
04079 #endif // platform
04080 #endif // DOCTEST_CONFIG_COLORS_WINDOWS && DOCTEST_CONFIG_COLORS_ANSI
04081 #endif // DOCTEST_CONFIG_COLORS_NONE
04082
04083 namespace doctest_detail_test_suite_ns {
04084 // holds the current test suite
04085 doctest::detail::TestSuite& getCurrentTestSuite() {
04086     static doctest::detail::TestSuite data{};
04087     return data;
04088 }
04089 } // namespace doctest_detail_test_suite_ns
04090
04091 namespace doctest {
04092 namespace {
04093     // the int (priority) is part of the key for automatic sorting - sadly one can register a
04094     // reporter with a duplicate name and a different priority but hopefully that won't happen often
04095     using reporterMap = std::map<std::pair<int, String>, reporterCreatorFunc>;
04096
04097     reporterMap& getReporters() {
04098         static reporterMap data;
04099         return data;
04100     }
04101     reporterMap& getListeners() {
04102         static reporterMap data;
04103         return data;
04104     }
04105 } // namespace
04106 namespace detail {
04107 #define DOCTEST_ITERATE_THROUGH_REPORTERS(function, ...) \
04108     for(auto& curr_rep : g_cs->reporters_currently_used) \
04109         curr_rep->function(__VA_ARGS__)
04110
04111     bool checkIfShouldThrow(assertType::Enum at) {
04112         if(at & assertType::is_require)
04113             return true;
04114
04115         if((at & assertType::is_check)
04116             && getContextOptions()->abort_after > 0 &&
04117             (g_cs->numAssertsFailed + g_cs->numAssertsFailedCurrentTest_atomic) >=
04118             getContextOptions()->abort_after)
04119             return true;
04120
04121         return false;
04122     }
04123
04124 #ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
04125     DOCTEST_NORETURN void throwException() {
04126         g_cs->shouldLogCurrentException = false;
04127         throw TestFailureException(); // NOLINT(hicpp-exception-baseclass)
04128     }
04129 #else // DOCTEST_CONFIG_NO_EXCEPTIONS
04130     void throwException() {}
04131 #endif // DOCTEST_CONFIG_NO_EXCEPTIONS
04132 } // namespace detail
04133
04134 namespace {
04135     using namespace detail;
04136     // matching of a string against a wildcard mask (case sensitivity configurable) taken from
04137     // https://www.codeproject.com/Articles/1088/Wildcard-string-compare-globbing
04138     int wilncmp(const char* str, const char* wild, bool caseSensitive) {
04139         const char* cp = str;
04140         const char* mp = wild;
04141
04142         while((*str) && (*wild != '*')) {
04143             if((caseSensitive ? (*wild != *str) : (tolower(*wild) != tolower(*str))) &&
04144                 (*wild != '?')) {
04145                 return 0;
04146             }
04147             wild++;
04148             str++;
04149         }
04150
04151         while(*str) {
04152             if(*wild == '*') {

```

```

04153         if(!+++wild) {
04154             return 1;
04155         }
04156         mp = wild;
04157         cp = str + 1;
04158     } else if((caseSensitive ? (*wild == *str) : (tolower(*wild) == tolower(*str))) ||
04159             (*wild == '?')) {
04160         wild++;
04161         str++;
04162     } else {
04163         wild = mp;
04164         str = cp++;
04165     }
04166 }
04167
04168 while(*wild == '*') {
04169     wild++;
04170 }
04171 return !*wild;
04172 }
04173
04174 // checks if the name matches any of the filters (and can be configured what to do when empty)
04175 bool matchesAny(const char* name, const std::vector<String>& filters, bool matchEmpty,
04176 bool caseSensitive) {
04177     if (filters.empty() && matchEmpty)
04178         return true;
04179     for (auto& curr : filters)
04180         if (wildcmp(name, curr.c_str(), caseSensitive))
04181             return true;
04182     return false;
04183 }
04184
04185 DOCTEST_NO_SANITIZE_INTEGER
04186 unsigned long long hash(unsigned long long a, unsigned long long b) {
04187     return (a << 5) + b;
04188 }
04189
04190 // C string hash function (djb2) - taken from http://www.cse.yorku.ca/~oz/hash.html
04191 DOCTEST_NO_SANITIZE_INTEGER
04192 unsigned long long hash(const char* str) {
04193     unsigned long long hash = 5381;
04194     char c;
04195     while ((c = *str++))
04196         hash = ((hash << 5) + hash) + c; // hash * 33 + c
04197     return hash;
04198 }
04199
04200 unsigned long long hash(const SubcaseSignature& sig) {
04201     return hash(hash(hash(sig.m_file), hash(sig.m_name.c_str())), sig.m_line);
04202 }
04203
04204 unsigned long long hash(const std::vector<SubcaseSignature>& sigs, size_t count) {
04205     unsigned long long running = 0;
04206     auto end = sigs.begin() + count;
04207     for (auto it = sigs.begin(); it != end; it++) {
04208         running = hash(running, hash(*it));
04209     }
04210     return running;
04211 }
04212
04213 unsigned long long hash(const std::vector<SubcaseSignature>& sigs) {
04214     unsigned long long running = 0;
04215     for (const SubcaseSignature& sig : sigs) {
04216         running = hash(running, hash(sig));
04217     }
04218     return running;
04219 }
04220 } // namespace
04221 namespace detail {
04222     bool Subcase::checkFilters() {
04223         if (g_cs->subcaseStack.size() < size_t(g_cs->subcase_filter_levels)) {
04224             if (!matchesAny(m_signature.m_name.c_str(), g_cs->filters[6], true, g_cs->case_sensitive))
04225                 return true;
04226             if (matchesAny(m_signature.m_name.c_str(), g_cs->filters[7], false, g_cs->case_sensitive))
04227                 return true;
04228         }
04229         return false;
04230     }
04231 }
04232
04233 Subcase::Subcase(const String& name, const char* file, int line)
04234     : m_signature({name, file, line}) {
04235     if (!g_cs->reachedLeaf) {
04236         if (g_cs->nextSubcaseStack.size() <= g_cs->subcaseStack.size()
04237             || g_cs->nextSubcaseStack[g_cs->subcaseStack.size()] == m_signature) {
04238             // Going down.
04239             if (checkFilters()) { return; }

```

```

04240         g_cs->subcaseStack.push_back(m_signature);
04241         g_cs->currentSubcaseDepth++;
04242         m_entered = true;
04243         DOCTEST_ITERATE_THROUGH_REPORTERS(subcase_start, m_signature);
04244     }
04245     } else {
04246         if (g_cs->subcaseStack[g_cs->currentSubcaseDepth] == m_signature) {
04247             // This subcase is reentered via control flow.
04248             g_cs->currentSubcaseDepth++;
04249             m_entered = true;
04250             DOCTEST_ITERATE_THROUGH_REPORTERS(subcase_start, m_signature);
04251         } else if (g_cs->nextSubcaseStack.size() <= g_cs->currentSubcaseDepth
04252             && g_cs->fullyTraversedSubcases.find(hash(hash(g_cs->subcaseStack,
04253 g_cs->currentSubcaseDepth), hash(m_signature)))
04254             == g_cs->fullyTraversedSubcases.end()) {
04255             if (checkFilters()) { return; }
04256             // This subcase is part of the one to be executed next.
04257             g_cs->nextSubcaseStack.clear();
04258             g_cs->nextSubcaseStack.insert(g_cs->nextSubcaseStack.end(),
04259                 g_cs->subcaseStack.begin(), g_cs->subcaseStack.begin() +
04260 g_cs->currentSubcaseDepth);
04261             g_cs->nextSubcaseStack.push_back(m_signature);
04262         }
04263     }
04264     DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4996) // std::uncaught_exception is deprecated in C++17
04265     DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH("-Wdeprecated-declarations")
04266     DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wdeprecated-declarations")
04267
04268     Subcase::~Subcase() {
04269         if (m_entered) {
04270             g_cs->currentSubcaseDepth--;
04271
04272             if (!g_cs->reachedLeaf) {
04273                 // Leaf.
04274                 g_cs->fullyTraversedSubcases.insert(hash(g_cs->subcaseStack));
04275                 g_cs->nextSubcaseStack.clear();
04276                 g_cs->reachedLeaf = true;
04277             } else if (g_cs->nextSubcaseStack.empty()) {
04278                 // All children are finished.
04279                 g_cs->fullyTraversedSubcases.insert(hash(g_cs->subcaseStack));
04280             }
04281         }
04282 #if defined(__cpp_lib_uncaught_exceptions) && __cpp_lib_uncaught_exceptions >= 201411L &&
04283     (!defined(__MAC_OS_X_VERSION_MIN_REQUIRED) || __MAC_OS_X_VERSION_MIN_REQUIRED >= 101200)
04284         if(std::uncaught_exceptions() > 0
04285 #else
04286         if(std::uncaught_exception()
04287 #endif
04288             && g_cs->shouldLogCurrentException) {
04289             DOCTEST_ITERATE_THROUGH_REPORTERS(
04290                 test_case_exception, {"exception thrown in subcase - will translate later "
04291                     "when the whole test case has been exited (cannot "
04292                     "translate while there is an active exception)",
04293                     false});
04294             g_cs->shouldLogCurrentException = false;
04295         }
04296         DOCTEST_ITERATE_THROUGH_REPORTERS(subcase_end, DOCTEST_EMPTY);
04297     }
04298 }
04299
04300 DOCTEST_CLANG_SUPPRESS_WARNING_POP
04301 DOCTEST_GCC_SUPPRESS_WARNING_POP
04302 DOCTEST_MSVC_SUPPRESS_WARNING_POP
04303
04304 Subcase::operator bool() const { return m_entered; }
04305
04306 Result::Result(bool passed, const String& decomposition)
04307     : m_passed(passed)
04308     , m_decomp(decomposition) {}
04309
04310 ExpressionComposer::ExpressionComposer(assertType::Enum at)
04311     : m_at(at) {}
04312
04313 TestSuite& TestSuite::operator*(const char* in) {
04314     m_test_suite = in;
04315     return *this;
04316 }
04317
04318 TestCase::TestCase(funcType test, const char* file, unsigned line, const TestSuite& test_suite,
04319     const String& type, int template_id) {
04320     m_file = file;
04321     m_line = line;
04322     m_name = nullptr; // will be later overridden in operator*
04323     m_test_suite = test_suite.m_test_suite;

```

```

04324         m_description      = test_suite.m_description;
04325         m_skip              = test_suite.m_skip;
04326         m_no_breaks         = test_suite.m_no_breaks;
04327         m_no_output         = test_suite.m_no_output;
04328         m_may_fail          = test_suite.m_may_fail;
04329         m_should_fail       = test_suite.m_should_fail;
04330         m_expected_failures = test_suite.m_expected_failures;
04331         m_timeout           = test_suite.m_timeout;
04332
04333         m_test              = test;
04334         m_type              = type;
04335         m_template_id       = template_id;
04336     }
04337
04338     TestCase::TestCase(const TestCase& other)
04339         : TestCaseData() {
04340         *this = other;
04341     }
04342
04343     DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(26434) // hides a non-virtual function
04344     TestCase& TestCase::operator=(const TestCase& other) {
04345         TestCaseData::operator=(other);
04346         m_test      = other.m_test;
04347         m_type       = other.m_type;
04348         m_template_id = other.m_template_id;
04349         m_full_name  = other.m_full_name;
04350
04351         if(m_template_id != -1)
04352             m_name = m_full_name.c_str();
04353         return *this;
04354     }
04355     DOCTEST_MSVC_SUPPRESS_WARNING_POP
04356
04357     TestCase& TestCase::operator*(const char* in) {
04358         m_name = in;
04359         // make a new name with an appended type for templated test case
04360         if(m_template_id != -1) {
04361             m_full_name = String(m_name) + "<" + m_type + ">";
04362             // redirect the name to point to the newly constructed full name
04363             m_name = m_full_name.c_str();
04364         }
04365         return *this;
04366     }
04367
04368     bool TestCase::operator<(const TestCase& other) const {
04369         // this will be used only to differentiate between test cases - not relevant for sorting
04370         if(m_line != other.m_line)
04371             return m_line < other.m_line;
04372         const int name_cmp = strcmp(m_name, other.m_name);
04373         if(name_cmp != 0)
04374             return name_cmp < 0;
04375         const int file_cmp = m_file.compare(other.m_file);
04376         if(file_cmp != 0)
04377             return file_cmp < 0;
04378         return m_template_id < other.m_template_id;
04379     }
04380
04381     // all the registered tests
04382     std::set<TestCase>& getRegisteredTests() {
04383         static std::set<TestCase> data;
04384         return data;
04385     }
04386 } // namespace detail
04387 namespace {
04388     using namespace detail;
04389     // for sorting tests by file/line
04390     bool fileOrderComparator(const TestCase* lhs, const TestCase* rhs) {
04391         // this is needed because MSVC gives different case for drive letters
04392         // for __FILE__ when evaluated in a header and a source file
04393         const int res = lhs->m_file.compare(rhs->m_file, bool(DOCTEST_MSVC));
04394         if(res != 0)
04395             return res < 0;
04396         if(lhs->m_line != rhs->m_line)
04397             return lhs->m_line < rhs->m_line;
04398         return lhs->m_template_id < rhs->m_template_id;
04399     }
04400
04401     // for sorting tests by suite/file/line
04402     bool suiteOrderComparator(const TestCase* lhs, const TestCase* rhs) {
04403         const int res = std::strcmp(lhs->m_test_suite, rhs->m_test_suite);
04404         if(res != 0)
04405             return res < 0;
04406         return fileOrderComparator(lhs, rhs);
04407     }
04408
04409     // for sorting tests by name/suite/file/line
04410     bool nameOrderComparator(const TestCase* lhs, const TestCase* rhs) {

```

```

04411     const int res = std::strcmp(lhs->m_name, rhs->m_name);
04412     if(res != 0)
04413         return res < 0;
04414     return suiteOrderComparator(lhs, rhs);
04415 }
04416
04417 DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wdeprecated-declarations")
04418 void color_to_stream(std::ostream& s, Color::Enum code) {
04419     static_cast<void>(s);    // for DOCTEST_CONFIG_COLORS_NONE or DOCTEST_CONFIG_COLORS_WINDOWS
04420     static_cast<void>(code); // for DOCTEST_CONFIG_COLORS_NONE
04421 #ifndef DOCTEST_CONFIG_COLORS_ANSI
04422     if(g_no_colors ||
04423         (isatty(STDOUT_FILENO) == false && getContextOptions()->force_colors == false))
04424         return;
04425
04426     auto col = "";
04427     // clang-format off
04428     switch(code) {
04429         case Color::Red:      col = "[0;31m"; break;
04430         case Color::Green:    col = "[0;32m"; break;
04431         case Color::Blue:     col = "[0;34m"; break;
04432         case Color::Cyan:     col = "[0;36m"; break;
04433         case Color::Yellow:   col = "[0;33m"; break;
04434         case Color::Grey:     col = "[1;30m"; break;
04435         case Color::LightGrey: col = "[0;37m"; break;
04436         case Color::BrightRed: col = "[1;31m"; break;
04437         case Color::BrightGreen: col = "[1;32m"; break;
04438         case Color::BrightWhite: col = "[1;37m"; break;
04439         case Color::Bright: // invalid
04440         case Color::None:
04441         case Color::White:
04442             default: col = "[0m";
04443     }
04444     // clang-format on
04445     s << "\033" << col;
04446 #endif // DOCTEST_CONFIG_COLORS_ANSI
04447
04448 #ifndef DOCTEST_CONFIG_COLORS_WINDOWS
04449     if(g_no_colors ||
04450         (_isatty(_fileno(stdout)) == false && getContextOptions()->force_colors == false))
04451         return;
04452
04453     static struct ConsoleHelper {
04454         HANDLE stdoutHandle;
04455         WORD origFgAttrs;
04456         WORD origBgAttrs;
04457
04458         ConsoleHelper() {
04459             stdoutHandle = GetStdHandle(STD_OUTPUT_HANDLE);
04460             CONSOLE_SCREEN_BUFFER_INFO csbiInfo;
04461             GetConsoleScreenBufferInfo(stdoutHandle, &csbiInfo);
04462             origFgAttrs = csbiInfo.wAttributes & ~(BACKGROUND_GREEN | BACKGROUND_RED |
04463                 BACKGROUND_BLUE | BACKGROUND_INTENSITY);
04464             origBgAttrs = csbiInfo.wAttributes & ~(FOREGROUND_GREEN | FOREGROUND_RED |
04465                 FOREGROUND_BLUE | FOREGROUND_INTENSITY);
04466         }
04467     } ch;
04468
04469 #define DOCTEST_SET_ATTR(x) SetConsoleTextAttribute(ch.stdoutHandle, x | ch.origBgAttrs)
04470
04471     // clang-format off
04472     switch (code) {
04473         case Color::White: DOCTEST_SET_ATTR(FOREGROUND_GREEN | FOREGROUND_RED |
04474             FOREGROUND_BLUE); break;
04475         case Color::Red: DOCTEST_SET_ATTR(FOREGROUND_RED);
04476         break;
04477         case Color::Green: DOCTEST_SET_ATTR(FOREGROUND_GREEN);
04478         break;
04479         case Color::Blue: DOCTEST_SET_ATTR(FOREGROUND_BLUE);
04480         break;
04481         case Color::Cyan: DOCTEST_SET_ATTR(FOREGROUND_BLUE | FOREGROUND_GREEN);
04482         break;
04483         case Color::Yellow: DOCTEST_SET_ATTR(FOREGROUND_RED | FOREGROUND_GREEN);
04484         break;
04485         case Color::Grey: DOCTEST_SET_ATTR(0);
04486         break;
04487         case Color::LightGrey: DOCTEST_SET_ATTR(FOREGROUND_INTENSITY);
04488         break;
04489         case Color::BrightRed: DOCTEST_SET_ATTR(FOREGROUND_INTENSITY | FOREGROUND_RED);
04490         break;
04491         case Color::BrightGreen: DOCTEST_SET_ATTR(FOREGROUND_INTENSITY | FOREGROUND_GREEN);
04492         break;
04493         case Color::BrightWhite: DOCTEST_SET_ATTR(FOREGROUND_INTENSITY | FOREGROUND_GREEN |
04494             FOREGROUND_RED | FOREGROUND_BLUE); break;
04495         case Color::None:
04496         case Color::Bright: // invalid
04497         default: DOCTEST_SET_ATTR(ch.origFgAttrs);

```

```

04487     }
04488     // clang-format on
04489 #endif // DOCTEST_CONFIG_COLORS_WINDOWS
04490 }
04491 DOCTEST_CLANG_SUPPRESS_WARNING_POP
04492
04493 std::vector<const IExceptionTranslator*> getExceptionTranslators() {
04494     static std::vector<const IExceptionTranslator*> data;
04495     return data;
04496 }
04497
04498 String translateActiveException() {
04499 #ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
04500     String res;
04501     auto& translators = getExceptionTranslators();
04502     for(auto& curr : translators)
04503         if(curr->translate(res))
04504             return res;
04505     // clang-format off
04506     DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH("-Wcatch-value")
04507     try {
04508         throw;
04509     } catch(std::exception& ex) {
04510         return ex.what();
04511     } catch(std::string& msg) {
04512         return msg.c_str();
04513     } catch(const char* msg) {
04514         return msg;
04515     } catch(...) {
04516         return "unknown exception";
04517     }
04518     DOCTEST_GCC_SUPPRESS_WARNING_POP
04519     // clang-format on
04520 #else // DOCTEST_CONFIG_NO_EXCEPTIONS
04521     return "";
04522 #endif // DOCTEST_CONFIG_NO_EXCEPTIONS
04523 }
04524 } // namespace
04525
04526 namespace detail {
04527     // used by the macros for registering tests
04528     int regTest(const TestCase& tc) {
04529         getRegisteredTests().insert(tc);
04530         return 0;
04531     }
04532
04533     // sets the current test suite
04534     int setTestSuite(const TestSuite& ts) {
04535         doctest_detail_test_suite_ns::getCurrentTestSuite() = ts;
04536         return 0;
04537     }
04538
04539 #ifndef DOCTEST_IS_DEBUGGER_ACTIVE
04540     bool isDebuggerActive() { return DOCTEST_IS_DEBUGGER_ACTIVE(); }
04541 #else // DOCTEST_IS_DEBUGGER_ACTIVE
04542 #ifndef DOCTEST_PLATFORM_LINUX
04543     class ErrnoGuard {
04544     public:
04545         ErrnoGuard() : m_oldErrno(errno) {}
04546         ~ErrnoGuard() { errno = m_oldErrno; }
04547     private:
04548         int m_oldErrno;
04549     };
04550     // See the comments in Catch2 for the reasoning behind this implementation:
04551     // https://github.com/catchorg/Catch2/blob/v2.13.1/include/internal/catch_debugger.cpp#L79-L102
04552     bool isDebuggerActive() {
04553         ErrnoGuard guard;
04554         std::ifstream in("/proc/self/status");
04555         for(std::string line; std::getline(in, line);) {
04556             static const int PREFIX_LEN = 11;
04557             if(line.compare(0, PREFIX_LEN, "TracerPid:") == 0) {
04558                 return line.length() > PREFIX_LEN && line[PREFIX_LEN] != '0';
04559             }
04560         }
04561         return false;
04562     }
04563 #elif defined(DOCTEST_PLATFORM_MAC)
04564     // The following function is taken directly from the following technical note:
04565     // https://developer.apple.com/library/archive/qa/qa1361/_index.html
04566     // Returns true if the current process is being debugged (either
04567     // running under the debugger or has a debugger attached post facto).
04568     bool isDebuggerActive() {
04569         int mib[4];
04570         kinfo_proc info;
04571         size_t size;
04572         // Initialize the flags so that, if sysctl fails for some bizarre
04573         // reason, we get a predictable result.

```



```

04574         info.kp_proc.p_flag = 0;
04575         // Initialize mib, which tells sysctl the info we want, in this case
04576         // we're looking for information about a specific process ID.
04577         mib[0] = CTL_KERN;
04578         mib[1] = KERN_PROC;
04579         mib[2] = KERN_PROC_PID;
04580         mib[3] = getpid();
04581         // Call sysctl.
04582         size = sizeof(info);
04583         if(sysctl(mib, DOCTEST_COUNTOF(mib), &info, &size, 0, 0) != 0) {
04584             std::cerr << "\nCall to sysctl failed - unable to determine if debugger is active **\n";
04585             return false;
04586         }
04587         // We're being debugged if the P_TRACED flag is set.
04588         return ((info.kp_proc.p_flag & P_TRACED) != 0);
04589     }
04590 #elif DOCTEST_MSVC || defined(__MINGW32__) || defined(__MINGW64__)
04591     bool isDebuggerActive() { return ::IsDebuggerPresent() != 0; }
04592 #else
04593     bool isDebuggerActive() { return false; }
04594 #endif // Platform
04595 #endif // DOCTEST_IS_DEBUGGER_ACTIVE
04596
04597     void registerExceptionTranslatorImpl(const IExceptionTranslator* et) {
04598         if(std::find(getExceptionTranslators().begin(), getExceptionTranslators().end(), et) ==
04599            getExceptionTranslators().end())
04600             getExceptionTranslators().push_back(et);
04601     }
04602
04603     DOCTEST_THREAD_LOCAL std::vector<IContextScope*> g_infoContexts; // for logging with INFO()
04604
04605     ContextScopeBase::ContextScopeBase() {
04606         g_infoContexts.push_back(this);
04607     }
04608
04609     ContextScopeBase::ContextScopeBase(ContextScopeBase&& other) noexcept {
04610         if (other.need_to_destroy) {
04611             other.destroy();
04612         }
04613         other.need_to_destroy = false;
04614         g_infoContexts.push_back(this);
04615     }
04616
04617     DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4996) // std::uncaught_exception is deprecated in C++17
04618     DOCTEST_GCC_SUPPRESS_WARNING_WITH_PUSH("-Wdeprecated-declarations")
04619     DOCTEST_CLANG_SUPPRESS_WARNING_WITH_PUSH("-Wdeprecated-declarations")
04620
04621     // destroy cannot be inlined into the destructor because that would mean calling stringify after
04622     // ContextScope has been destroyed (base class destructors run after derived class destructors).
04623     // Instead, ContextScope calls this method directly from its destructor.
04624     void ContextScopeBase::destroy() {
04625 #if defined(__cpp_lib_uncaught_exceptions) && __cpp_lib_uncaught_exceptions >= 201411L &&
04626     (!defined(__MAC_OS_X_VERSION_MIN_REQUIRED) || __MAC_OS_X_VERSION_MIN_REQUIRED >= 101200)
04627         if(std::uncaught_exceptions() > 0) {
04628 #else
04629         if(std::uncaught_exception()) {
04630 #endif
04631             std::ostringstream s;
04632             this->stringify(&s);
04633             g_cs->stringifiedContexts.push_back(s.str().c_str());
04634         }
04635         g_infoContexts.pop_back();
04636     }
04637
04638     DOCTEST_CLANG_SUPPRESS_WARNING_POP
04639     DOCTEST_GCC_SUPPRESS_WARNING_POP
04640     DOCTEST_MSVC_SUPPRESS_WARNING_POP
04641 } // namespace detail
04642 namespace {
04643     using namespace detail;
04644
04645     #if !defined(DOCTEST_CONFIG_POSIX_SIGNALS) && !defined(DOCTEST_CONFIG_WINDOWS_SEH)
04646     struct FatalConditionHandler
04647     {
04648         static void reset() {}
04649         static void allocateAltStackMem() {}
04650         static void freeAltStackMem() {}
04651     };
04652     #else // DOCTEST_CONFIG_POSIX_SIGNALS || DOCTEST_CONFIG_WINDOWS_SEH
04653     void reportFatal(const std::string&);
04654
04655     #ifdef DOCTEST_PLATFORM_WINDOWS
04656     struct SignalDefs
04657     {
04658         {
04659             DWORD id;

```

```

04660     const char* name;
04661 };
04662 // There is no 1-1 mapping between signals and windows exceptions.
04663 // Windows can easily distinguish between SO and SigSegV,
04664 // but SigInt, SigTerm, etc are handled differently.
04665 SignalDefs signalDefs[] = {
04666     {static_cast<DWORD>(EXCEPTION_ILLEGAL_INSTRUCTION),
04667      "SIGILL - Illegal instruction signal"},
04668     {static_cast<DWORD>(EXCEPTION_STACK_OVERFLOW), "SIGSEGV - Stack overflow"},
04669     {static_cast<DWORD>(EXCEPTION_ACCESS_VIOLATION),
04670      "SIGSEGV - Segmentation violation signal"},
04671     {static_cast<DWORD>(EXCEPTION_INT_DIVIDE_BY_ZERO), "Divide by zero error"},
04672 };
04673
04674 struct FatalConditionHandler
04675 {
04676     static LONG CALLBACK handleException(PEXCEPTION_POINTERS ExceptionInfo) {
04677         // Multiple threads may enter this filter/handler at once. We want the error message to be
04678         // console just once no matter how many threads have crashed.
04679         DOCTEST_DECLARE_STATIC_MUTEX(mutex)
04680         static bool execute = true;
04681         {
04682             DOCTEST_LOCK_MUTEX(mutex)
04683             if(execute) {
04684                 bool reported = false;
04685                 for(size_t i = 0; i < DOCTEST_COUNTOF(signalDefs); ++i) {
04686                     if(ExceptionInfo->ExceptionRecord->ExceptionCode == signalDefs[i].id) {
04687                         reportFatal(signalDefs[i].name);
04688                         reported = true;
04689                         break;
04690                     }
04691                 }
04692                 if(reported == false)
04693                     reportFatal("Unhandled SEH exception caught");
04694                 if(isDebuggerActive() && !g_cs->no_breaks)
04695                     DOCTEST_BREAK_INTO_DEBUGGER();
04696             }
04697             execute = false;
04698         }
04699         std::exit(EXIT_FAILURE);
04700     }
04701
04702     static void allocateAltStackMem() {}
04703     static void freeAltStackMem() {}
04704
04705     FatalConditionHandler() {
04706         isSet = true;
04707         // 32k seems enough for doctest to handle stack overflow,
04708         // but the value was found experimentally, so there is no strong guarantee
04709         guaranteeSize = 32 * 1024;
04710         // Register an unhandled exception filter
04711         previousTop = SetUnhandledExceptionFilter(handleException);
04712         // Pass in guarantee size to be filled
04713         SetThreadStackGuarantee(&guaranteeSize);
04714
04715         // On Windows uncaught exceptions from another thread, exceptions from
04716         // destructors, or calls to std::terminate are not a SEH exception
04717
04718         // The terminal handler gets called when:
04719         // - std::terminate is called FROM THE TEST RUNNER THREAD
04720         // - an exception is thrown from a destructor FROM THE TEST RUNNER THREAD
04721         original_terminate_handler = std::get_terminate();
04722         std::set_terminate([]() DOCTEST_NOEXCEPT {
04723             reportFatal("Terminate handler called");
04724             if(isDebuggerActive() && !g_cs->no_breaks)
04725                 DOCTEST_BREAK_INTO_DEBUGGER();
04726             std::exit(EXIT_FAILURE); // explicitly exit - otherwise the SIGABRT handler may be
04727             // called as well
04728         });
04729
04730         // SIGABRT is raised when:
04731         // - std::terminate is called FROM A DIFFERENT THREAD
04732         // - an exception is thrown from a destructor FROM A DIFFERENT THREAD
04733         // - an uncaught exception is thrown FROM A DIFFERENT THREAD
04734         prev_sigabrt_handler = std::signal(SIGABRT, [](int signal) DOCTEST_NOEXCEPT {
04735             if(signal == SIGABRT) {
04736                 reportFatal("SIGABRT - Abort (abnormal termination) signal");
04737                 if(isDebuggerActive() && !g_cs->no_breaks)
04738                     DOCTEST_BREAK_INTO_DEBUGGER();
04739                 std::exit(EXIT_FAILURE);
04740             }
04741         });
04742
04743         // The following settings are taken from google test, and more
04744         // specifically from UnitTest::Run() inside of gtest.cc

```

```

04745         // the user does not want to see pop-up dialogs about crashes
04746         prev_error_mode_1 = SetErrorMode(SEM_FAILCRITICALERRORS | SEM_NOALIGNMENTFAULTEXCEPT |
04747             SEM_NOGPFAULTERRORBOX | SEM_NOOPENFILEERRORBOX);
04748         // This forces the abort message to go to stderr in all circumstances.
04749         prev_error_mode_2 = _set_error_mode(_OUT_TO_STDERR);
04750         // In the debug version, Visual Studio pops up a separate dialog
04751         // offering a choice to debug the aborted program - we want to disable that.
04752         prev_abort_behavior = _set_abort_behavior(0x0, _WRITE_ABORT_MSG | _CALL_REPORTFAULT);
04753         // In debug mode, the Windows CRT can crash with an assertion over invalid
04754         // input (e.g. passing an invalid file descriptor). The default handling
04755         // for these assertions is to pop up a dialog and wait for user input.
04756         // Instead ask the CRT to dump such assertions to stderr non-interactively.
04757         prev_report_mode = _CrtSetReportMode(_CRT_ASSERT, _CRTDBG_MODE_FILE | _CRTDBG_MODE_DEBUG);
04758         prev_report_file = _CrtSetReportFile(_CRT_ASSERT, _CRTDBG_FILE_STDERR);
04759     }
04760
04761     static void reset() {
04762         if(isSet) {
04763             // Unregister handler and restore the old guarantee
04764             SetUnhandledExceptionFilter(previousTop);
04765             SetThreadStackGuarantee(&guaranteeSize);
04766             std::set_terminate(original_terminate_handler);
04767             std::signal(SIGABRT, prev_sigabrt_handler);
04768             SetErrorMode(prev_error_mode_1);
04769             _set_error_mode(prev_error_mode_2);
04770             _set_abort_behavior(prev_abort_behavior, _WRITE_ABORT_MSG | _CALL_REPORTFAULT);
04771             static_cast<void>(_CrtSetReportMode(_CRT_ASSERT, prev_report_mode));
04772             static_cast<void>(_CrtSetReportFile(_CRT_ASSERT, prev_report_file));
04773             isSet = false;
04774         }
04775     }
04776
04777     ~FatalConditionHandler() { reset(); }
04778
04779 private:
04780     static UINT          prev_error_mode_1;
04781     static int           prev_error_mode_2;
04782     static unsigned int  prev_abort_behavior;
04783     static int           prev_report_mode;
04784     static _HFILE        prev_report_file;
04785     static void (DOCTEST_CDECL *prev_sigabrt_handler)(int);
04786     static std::terminate_handler original_terminate_handler;
04787     static bool isSet;
04788     static ULONG guaranteeSize;
04789     static LPTOP_LEVEL_EXCEPTION_FILTER previousTop;
04790 };
04791
04792     UINT          FatalConditionHandler::prev_error_mode_1;
04793     int           FatalConditionHandler::prev_error_mode_2;
04794     unsigned int  FatalConditionHandler::prev_abort_behavior;
04795     int           FatalConditionHandler::prev_report_mode;
04796     _HFILE        FatalConditionHandler::prev_report_file;
04797     void (DOCTEST_CDECL *FatalConditionHandler::prev_sigabrt_handler)(int);
04798     std::terminate_handler FatalConditionHandler::original_terminate_handler;
04799     bool FatalConditionHandler::isSet = false;
04800     ULONG FatalConditionHandler::guaranteeSize = 0;
04801     LPTOP_LEVEL_EXCEPTION_FILTER FatalConditionHandler::previousTop = nullptr;
04802
04803 #else // DOCTEST_PLATFORM_WINDOWS
04804
04805     struct SignalDefs
04806     {
04807         int      id;
04808         const char* name;
04809     };
04810     SignalDefs signalDefs[] = {{SIGINT, "SIGINT - Terminal interrupt signal"},
04811         {SIGILL, "SIGILL - Illegal instruction signal"},
04812         {SIGFPE, "SIGFPE - Floating point error signal"},
04813         {SIGSEGV, "SIGSEGV - Segmentation violation signal"},
04814         {SIGTERM, "SIGTERM - Termination request signal"},
04815         {SIGABRT, "SIGABRT - Abort (abnormal termination) signal"};
04816
04817     struct FatalConditionHandler
04818     {
04819         static bool          isSet;
04820         static struct sigaction oldSigActions[DOCTEST_COUNTOF(signalDefs)];
04821         static stack_t        oldSigStack;
04822         static size_t         altStackSize;
04823         static char*          altStackMem;
04824
04825         static void handleSignal(int sig) {
04826             const char* name = "<unknown signal>";
04827             for(std::size_t i = 0; i < DOCTEST_COUNTOF(signalDefs); ++i) {
04828                 SignalDefs& def = signalDefs[i];
04829                 if(sig == def.id) {
04830                     name = def.name;
04831                     break;

```

```

04832         }
04833     }
04834     reset();
04835     reportFatal(name);
04836     raise(sig);
04837 }
04838
04839 static void allocateAltStackMem() {
04840     altStackMem = new char[altStackSize];
04841 }
04842
04843 static void freeAltStackMem() {
04844     delete[] altStackMem;
04845 }
04846
04847 FatalConditionHandler() {
04848     isSet = true;
04849     stack_t sigStack;
04850     sigStack.ss_sp = altStackMem;
04851     sigStack.ss_size = altStackSize;
04852     sigStack.ss_flags = 0;
04853     sigaltstack(&sigStack, &oldSigStack);
04854     struct sigaction sa = {};
04855     sa.sa_handler = handleSignal;
04856     sa.sa_flags = SA_ONSTACK;
04857     for(std::size_t i = 0; i < DOCTEST_COUNTOF(signalDefs); ++i) {
04858         sigaction(signalDefs[i].id, &sa, &oldSigActions[i]);
04859     }
04860 }
04861
04862 ~FatalConditionHandler() { reset(); }
04863 static void reset() {
04864     if(isSet) {
04865         // Set signals back to previous values -- hopefully nobody overwrote them in the
04866         meantime
04867         for(std::size_t i = 0; i < DOCTEST_COUNTOF(signalDefs); ++i) {
04868             sigaction(signalDefs[i].id, &oldSigActions[i], nullptr);
04869         }
04870         // Return the old stack
04871         sigaltstack(&oldSigStack, nullptr);
04872         isSet = false;
04873     }
04874 };
04875
04876 bool FatalConditionHandler::isSet = false;
04877 struct sigaction FatalConditionHandler::oldSigActions[DOCTEST_COUNTOF(signalDefs)] = {};
04878 stack_t FatalConditionHandler::oldSigStack = {};
04879 size_t FatalConditionHandler::altStackSize = 4 * SIGSTKSZ;
04880 char* FatalConditionHandler::altStackMem = nullptr;
04881
04882 #endif // DOCTEST_PLATFORM_WINDOWS
04883 #endif // DOCTEST_CONFIG_POSIX_SIGNALS || DOCTEST_CONFIG_WINDOWS_SEH
04884
04885 } // namespace
04886
04887 namespace {
04888     using namespace detail;
04889
04890 #ifdef DOCTEST_PLATFORM_WINDOWS
04891 #define DOCTEST_OUTPUT_DEBUG_STRING(text) ::OutputDebugStringA(text)
04892 #else
04893     // TODO: integration with XCode and other IDEs
04894 #define DOCTEST_OUTPUT_DEBUG_STRING(text)
04895 #endif // Platform
04896
04897 void addAssert(assertType::Enum at) {
04898     if((at & assertType::is_warn) == 0)
04899         g_cs->numAssertsCurrentTest_atomic++;
04900 }
04901
04902 void addFailedAssert(assertType::Enum at) {
04903     if((at & assertType::is_warn) == 0)
04904         g_cs->numAssertsFailedCurrentTest_atomic++;
04905 }
04906
04907 #if defined(DOCTEST_CONFIG_POSIX_SIGNALS) || defined(DOCTEST_CONFIG_WINDOWS_SEH)
04908 void reportFatal(const std::string& message) {
04909     g_cs->failure_flags |= TestCaseFailureReason::Crash;
04910
04911     DOCTEST_ITERATE_THROUGH_REPORTERS(test_case_exception, {message.c_str(), true});
04912
04913     while (g_cs->subcaseStack.size()) {
04914         g_cs->subcaseStack.pop_back();
04915         DOCTEST_ITERATE_THROUGH_REPORTERS(subcase_end, DOCTEST_EMPTY);
04916     }
04917 }

```

```

04918         g_cs->finalizeTestCaseData();
04919
04920         DOCTEST_ITERATE_THROUGH_REPORTERS(test_case_end, *g_cs);
04921
04922         DOCTEST_ITERATE_THROUGH_REPORTERS(test_run_end, *g_cs);
04923     }
04924 #endif // DOCTEST_CONFIG_POSIX_SIGNALS || DOCTEST_CONFIG_WINDOWS_SEH
04925 } // namespace
04926
04927 AssertData::AssertData(assertType::Enum at, const char* file, int line, const char* expr,
04928     const char* exception_type, const StringContains& exception_string)
04929 : m_test_case(g_cs->currentTest), m_at(at), m_file(file), m_line(line), m_expr(expr),
04930     m_failed(true), m_threw(false), m_threw_as(false), m_exception_type(exception_type),
04931     m_exception_string(exception_string) {
04932 #if DOCTEST_MSVC
04933     if (m_expr[0] == ' ') // this happens when variadic macros are disabled under MSVC
04934         ++m_expr;
04935 #endif // MSVC
04936 }
04937
04938 namespace detail {
04939     ResultBuilder::ResultBuilder(assertType::Enum at, const char* file, int line, const char* expr,
04940         const char* exception_type, const String& exception_string)
04941         : AssertData(at, file, line, expr, exception_type, exception_string) {}
04942
04943     ResultBuilder::ResultBuilder(assertType::Enum at, const char* file, int line, const char* expr,
04944         const char* exception_type, const Contains& exception_string)
04945         : AssertData(at, file, line, expr, exception_type, exception_string) {}
04946
04947     void ResultBuilder::setResult(const Result& res) {
04948         m_decomp = res.m_decomp;
04949         m_failed = !res.m_passed;
04950     }
04951
04952     void ResultBuilder::translateException() {
04953         m_threw = true;
04954         m_exception = translateActiveException();
04955     }
04956
04957     bool ResultBuilder::log() {
04958         if(m_at & assertType::is_throws) {
04959             m_failed = !m_threw;
04960         } else if((m_at & assertType::is_throws_as) && (m_at & assertType::is_throws_with)) {
04961             m_failed = !m_threw_as || !m_exception_string.check(m_exception);
04962         } else if(m_at & assertType::is_throws_as) {
04963             m_failed = !m_threw_as;
04964         } else if(m_at & assertType::is_throws_with) {
04965             m_failed = !m_exception_string.check(m_exception);
04966         } else if(m_at & assertType::is_nothrow) {
04967             m_failed = m_threw;
04968         }
04969
04970         if(m_exception.size())
04971             m_exception = "\"" + m_exception + "\"";
04972
04973         if(is_running_in_test) {
04974             addAssert(m_at);
04975             DOCTEST_ITERATE_THROUGH_REPORTERS(log_assert, *this);
04976
04977             if(m_failed)
04978                 addFailedAssert(m_at);
04979         } else if(m_failed) {
04980             failed_out_of_a_testing_context(*this);
04981         }
04982
04983         return m_failed && isDebuggerActive() && !getContextOptions()->no_breaks &&
04984             (g_cs->currentTest == nullptr || !g_cs->currentTest->m_no_breaks); // break into debugger
04985     }
04986
04987     void ResultBuilder::react() const {
04988         if(m_failed && checkIfShouldThrow(m_at))
04989             throwException();
04990     }
04991
04992     void failed_out_of_a_testing_context(const AssertData& ad) {
04993         if(g_cs->ah)
04994             g_cs->ah(ad);
04995         else
04996             std::abort();
04997     }
04998
04999     bool decomp_assert(assertType::Enum at, const char* file, int line, const char* expr,
05000         const Result& result) {
05001         bool failed = !result.m_passed;
05002
05003         // #####
05004         // IF THE DEBUGGER BREAKS HERE - GO 1 LEVEL UP IN THE CALLSTACK FOR THE FAILING ASSERT

```

```

05005         // THIS IS THE EFFECT OF HAVING 'DOCTEST_CONFIG_SUPER_FAST_ASSERTS' DEFINED
05006         // #####
05007         DOCTEST_ASSERT_OUT_OF_TESTS(result.m_decomp);
05008         DOCTEST_ASSERT_IN_TESTS(result.m_decomp);
05009         return !failed;
05010     }
05011
05012     MessageBuilder::MessageBuilder(const char* file, int line, assertType::Enum severity) {
05013         m_stream = tlssPush();
05014         m_file = file;
05015         m_line = line;
05016         m_severity = severity;
05017     }
05018
05019     MessageBuilder::~MessageBuilder() {
05020         if (!logged)
05021             tlssPop();
05022     }
05023
05024     DOCTEST_DEFINE_INTERFACE(IEExceptionTranslator)
05025
05026     bool MessageBuilder::log() {
05027         if (!logged) {
05028             m_string = tlssPop();
05029             logged = true;
05030         }
05031
05032         DOCTEST_ITERATE_THROUGH_REPORTERS(log_message, *this);
05033
05034         const bool isWarn = m_severity & assertType::is_warn;
05035
05036         // warn is just a message in this context so we don't treat it as an assert
05037         if(!isWarn) {
05038             addAssert(m_severity);
05039             addFailedAssert(m_severity);
05040         }
05041
05042         return isDebuggerActive() && !getContextOptions()->no_breaks && !isWarn &&
05043             (g_cs->currentTest == nullptr || !g_cs->currentTest->m_no_breaks); // break into debugger
05044     }
05045
05046     void MessageBuilder::react() {
05047         if(m_severity & assertType::is_require)
05048             throwException();
05049     }
05050 } // namespace detail
05051 namespace {
05052     using namespace detail;
05053
05054     // clang-format off
05055
05056     // =====
05057     // The following code has been taken verbatim from Catch2/include/internal/catch_xmlwriter.h/cpp
05058     // This is done so cherry-picking bug fixes is trivial - even the style/formatting is untouched.
05059     // =====
05060
05061     class XmlEncode {
05062     public:
05063         enum ForWhat { ForTextNodes, ForAttributes };
05064
05065         XmlEncode( std::string const& str, ForWhat forWhat = ForTextNodes );
05066
05067         void encodeTo( std::ostream& os ) const;
05068
05069         friend std::ostream& operator << ( std::ostream& os, XmlEncode const& xmlEncode );
05070
05071     private:
05072         std::string m_str;
05073         ForWhat m_forWhat;
05074     };
05075
05076     class XmlWriter {
05077     public:
05078         class ScopedElement {
05079         public:
05080             ScopedElement( XmlWriter* writer );
05081
05082             ScopedElement( ScopedElement&& other ) DOCTEST_NOEXCEPT;
05083             ScopedElement& operator=( ScopedElement&& other ) DOCTEST_NOEXCEPT;
05084
05085             ~ScopedElement();
05086
05087             ScopedElement& writeText( std::string const& text, bool indent = true );
05088
05089             template<typename T>
05090             ScopedElement& writeAttribute( std::string const& name, T const& attribute ) {

```

```

05092         m_writer->writeAttribute( name, attribute );
05093         return *this;
05094     }
05095
05096     private:
05097         mutable XmlWriter* m_writer = nullptr;
05098     };
05099
05100 #ifndef DOCTEST_CONFIG_NO_INCLUDE_Iostream
05101     XmlWriter( std::ostream& os = std::cout );
05102 #else // DOCTEST_CONFIG_NO_INCLUDE_Iostream
05103     XmlWriter( std::ostream& os );
05104 #endif // DOCTEST_CONFIG_NO_INCLUDE_Iostream
05105     ~XmlWriter();
05106
05107     XmlWriter( XmlWriter const& ) = delete;
05108     XmlWriter& operator=( XmlWriter const& ) = delete;
05109
05110     XmlWriter& startElement( std::string const& name );
05111
05112     ScopedElement scopedElement( std::string const& name );
05113
05114     XmlWriter& endElement();
05115
05116     XmlWriter& writeAttribute( std::string const& name, std::string const& attribute );
05117
05118     XmlWriter& writeAttribute( std::string const& name, const char* attribute );
05119
05120     XmlWriter& writeAttribute( std::string const& name, bool attribute );
05121
05122     template<typename T>
05123     XmlWriter& writeAttribute( std::string const& name, T const& attribute ) {
05124         std::stringstream rss;
05125         rss << attribute;
05126         return writeAttribute( name, rss.str() );
05127     }
05128
05129     XmlWriter& writeText( std::string const& text, bool indent = true );
05130
05131     //XmlWriter& writeComment( std::string const& text );
05132
05133     //void writeStylesheetRef( std::string const& url );
05134
05135     //XmlWriter& writeBlankLine();
05136
05137     void ensureTagClosed();
05138
05139     void writeDeclaration();
05140
05141     private:
05142
05143         void newlineIfNecessary();
05144
05145         bool m_tagIsOpen = false;
05146         bool m_needsNewline = false;
05147         std::vector<std::string> m_tags;
05148         std::string m_indent;
05149         std::ostream& m_os;
05150     };
05151
05152 // =====
05153 // The following code has been taken verbatim from Catch2/include/internal/catch_xmlwriter.h/cpp
05154 // This is done so cherry-picking bug fixes is trivial - even the style/formatting is untouched.
05155 // =====
05156
05157 using uchar = unsigned char;
05158
05159 namespace {
05160
05161     size_t trailingBytes(unsigned char c) {
05162         if ((c & 0xE0) == 0xC0) {
05163             return 2;
05164         }
05165         if ((c & 0xF0) == 0xE0) {
05166             return 3;
05167         }
05168         if ((c & 0xF8) == 0xF0) {
05169             return 4;
05170         }
05171         DOCTEST_INTERNAL_ERROR("Invalid multibyte utf-8 start byte encountered");
05172     }
05173
05174     uint32_t headerValue(unsigned char c) {
05175         if ((c & 0xE0) == 0xC0) {
05176             return c & 0x1F;
05177         }
05178         if ((c & 0xF0) == 0xE0) {

```

```

05179         return c & 0x0F;
05180     }
05181     if ((c & 0xF8) == 0xF0) {
05182         return c & 0x07;
05183     }
05184     DOCTEST_INTERNAL_ERROR("Invalid multibyte utf-8 start byte encountered");
05185 }
05186
05187 void hexEscapeChar(std::ostream& os, unsigned char c) {
05188     std::ios_base::fmtflags f(os.flags());
05189     os << "\\x"
05190         << std::uppercase << std::hex << std::setfill('0') << std::setw(2)
05191         << static_cast<int>(c);
05192     os.flags(f);
05193 }
05194
05195 } // anonymous namespace
05196
05197 XmlEncode::XmlEncode( std::string const& str, ForWhat forWhat )
05198 :   m_str( str ),
05199   m_forWhat( forWhat )
05200 {}
05201
05202 void XmlEncode::encodeTo( std::ostream& os ) const {
05203     // Apostrophe escaping not necessary if we always use " to write attributes
05204     // (see: https://www.w3.org/TR/xml/#syntax)
05205
05206     for( std::size_t idx = 0; idx < m_str.size(); ++ idx ) {
05207         uchar c = m_str[idx];
05208         switch (c) {
05209             case '<':   os << "&lt;"; break;
05210             case '&':   os << "&amp;"; break;
05211
05212             case '>':
05213                 // See: https://www.w3.org/TR/xml/#syntax
05214                 if (idx > 2 && m_str[idx - 1] == '[' && m_str[idx - 2] == '[')
05215                     os << "&gt;";
05216                 else
05217                     os << c;
05218                 break;
05219
05220             case '"':
05221                 if (m_forWhat == ForAttributes)
05222                     os << "&quot;";
05223                 else
05224                     os << c;
05225                 break;
05226
05227             default:
05228                 // Check for control characters and invalid utf-8
05229
05230                 // Escape control characters in standard ascii
05231                 // see
05232                 https://stackoverflow.com/questions/404107/why-are-control-characters-illegal-in-xml-1-0
05233                 if (c < 0x09 || (c > 0x0D && c < 0x20) || c == 0x7F) {
05234                     hexEscapeChar(os, c);
05235                     break;
05236
05237                     // Plain ASCII: Write it to stream
05238                     if (c < 0x7F) {
05239                         os << c;
05240                         break;
05241                     }
05242
05243                     // UTF-8 territory
05244                     // Check if the encoding is valid and if it is not, hex escape bytes.
05245                     // Important: We do not check the exact decoded values for validity, only the encoding
05246
05247                     format
05248                     // First check that this bytes is a valid lead byte:
05249                     // This means that it is not encoded as 1111 1XXX
05250                     // Or as 10XX XXXX
05251                     if (c < 0xC0 ||
05252                         c >= 0xF8) {
05253                         hexEscapeChar(os, c);
05254                         break;
05255                     }
05256
05257                     auto encBytes = trailingBytes(c);
05258                     // Are there enough bytes left to avoid accessing out-of-bounds memory?
05259                     if (idx + encBytes - 1 >= m_str.size()) {
05260                         hexEscapeChar(os, c);
05261                         break;
05262                     }
05263                     // The header is valid, check data
05264                     // The next encBytes bytes must together be a valid utf-8
05265                     // This means: bitpattern 10XX XXXX and the extracted value is sane (ish)

```



```

05264         bool valid = true;
05265         uint32_t value = headerValue(c);
05266         for (std::size_t n = 1; n < encBytes; ++n) {
05267             uchar nc = m_str[idx + n];
05268             valid &= ((nc & 0xC0) == 0x80);
05269             value = (value << 6) | (nc & 0x3F);
05270         }
05271
05272         if (
05273             // Wrong bit pattern of following bytes
05274             (!valid) ||
05275             // Overlong encodings
05276             (value < 0x80) ||
05277             (
05278                 value < 0x800    && encBytes > 2) || // removed "0x80 <= value
05279             &&" because redundant
05280             (0x800 < value && value < 0x10000 && encBytes > 3) ||
05281             // Encoded value out of range
05282             (value >= 0x110000)
05283             ) {
05284             hexEscapeChar(os, c);
05285             break;
05286         }
05287
05288         // If we got here, this is in fact a valid(ish) utf-8 sequence
05289         for (std::size_t n = 0; n < encBytes; ++n) {
05290             os << m_str[idx + n];
05291         }
05292         idx += encBytes - 1;
05293         break;
05294     }
05295 }
05296
05297 std::ostream& operator << ( std::ostream& os, XmlEncode const& xmlEncode ) {
05298     xmlEncode.encodeTo( os );
05299     return os;
05300 }
05301
05302 XmlWriter::ScopedElement::ScopedElement( XmlWriter* writer )
05303 :   m_writer( writer )
05304 {}
05305
05306 XmlWriter::ScopedElement::ScopedElement( ScopedElement&& other ) DOCTEST_NOEXCEPT
05307 :   m_writer( other.m_writer ){
05308     other.m_writer = nullptr;
05309 }
05310
05311 XmlWriter::ScopedElement& XmlWriter::ScopedElement::operator=( ScopedElement&& other )
05312 DOCTEST_NOEXCEPT {
05313     if ( m_writer ) {
05314         m_writer->endElement();
05315     }
05316     m_writer = other.m_writer;
05317     other.m_writer = nullptr;
05318     return *this;
05319 }
05320
05321 XmlWriter::ScopedElement::~~ScopedElement() {
05322     if( m_writer )
05323         m_writer->endElement();
05324 }
05325
05326 XmlWriter::ScopedElement& XmlWriter::ScopedElement::writeText( std::string const& text, bool
05327 indent ) {
05328     m_writer->writeText( text, indent );
05329     return *this;
05330 }
05331
05332 XmlWriter::XmlWriter( std::ostream& os ) : m_os( os )
05333 {}
05334
05335 // writeDeclaration(); // called explicitly by the reporters that use the writer class - see
05336 issue #627
05337 }
05338
05339 XmlWriter::~~XmlWriter() {
05340     while( !m_tags.empty() )
05341         endElement();
05342 }
05343
05344 XmlWriter& XmlWriter::startElement( std::string const& name ) {
05345     ensureTagClosed();
05346     newlineIfNecessary();
05347     m_os << m_indent << '<' << name;
05348     m_tags.push_back( name );
05349     m_indent += "  ";
05350     m_tagIsOpen = true;
05351     return *this;
05352 }

```

```

05347     }
05348
05349     XmlWriter::ScopedElement XmlWriter::scopedElement( std::string const& name ) {
05350         ScopedElement scoped( this );
05351         startElement( name );
05352         return scoped;
05353     }
05354
05355     XmlWriter& XmlWriter::endElement() {
05356         newlineIfNecessary();
05357         m_indent = m_indent.substr( 0, m_indent.size()-2 );
05358         if( m_tagIsOpen ) {
05359             m_os << ">";
05360             m_tagIsOpen = false;
05361         }
05362         else {
05363             m_os << m_indent << "</" << m_tags.back() << ">";
05364         }
05365         m_os << std::endl;
05366         m_tags.pop_back();
05367         return *this;
05368     }
05369
05370     XmlWriter& XmlWriter::writeAttribute( std::string const& name, std::string const& attribute ) {
05371         if( !name.empty() && !attribute.empty() )
05372             m_os << ' ' << name << "=" << XmlEncode( attribute, XmlEncode::ForAttributes ) << "'";
05373         return *this;
05374     }
05375
05376     XmlWriter& XmlWriter::writeAttribute( std::string const& name, const char* attribute ) {
05377         if( !name.empty() && attribute && attribute[0] != '\0' )
05378             m_os << ' ' << name << "=" << XmlEncode( attribute, XmlEncode::ForAttributes ) << "'";
05379         return *this;
05380     }
05381
05382     XmlWriter& XmlWriter::writeAttribute( std::string const& name, bool attribute ) {
05383         m_os << ' ' << name << "=" << ( attribute ? "true" : "false" ) << "'";
05384         return *this;
05385     }
05386
05387     XmlWriter& XmlWriter::writeText( std::string const& text, bool indent ) {
05388         if( !text.empty() ){
05389             bool tagWasOpen = m_tagIsOpen;
05390             ensureTagClosed();
05391             if( tagWasOpen && indent )
05392                 m_os << m_indent;
05393             m_os << XmlEncode( text );
05394             m_needsNewline = true;
05395         }
05396         return *this;
05397     }
05398
05399     //XmlWriter& XmlWriter::writeComment( std::string const& text ) {
05400     //    ensureTagClosed();
05401     //    m_os << m_indent << "<!--" << text << "-->";
05402     //    m_needsNewline = true;
05403     //    return *this;
05404     //}
05405
05406     //void XmlWriter::writeStylesheetRef( std::string const& url ) {
05407     //    m_os << "<?xml-stylesheet type=\"text/xsl\" href=\"" << url << "\"?>\n";
05408     //}
05409
05410     //XmlWriter& XmlWriter::writeBlankLine() {
05411     //    ensureTagClosed();
05412     //    m_os << '\n';
05413     //    return *this;
05414     //}
05415
05416     void XmlWriter::ensureTagClosed() {
05417         if( m_tagIsOpen ) {
05418             m_os << ">" << std::endl;
05419             m_tagIsOpen = false;
05420         }
05421     }
05422
05423     void XmlWriter::writeDeclaration() {
05424         m_os << "<?xml version=\"1.0\" encoding=\"UTF-8\"?>\n";
05425     }
05426
05427     void XmlWriter::newlineIfNecessary() {
05428         if( m_needsNewline ) {
05429             m_os << std::endl;
05430             m_needsNewline = false;
05431         }
05432     }
05433

```

```

05434 // =====
05435 // End of copy-pasted code from Catch
05436 // =====
05437
05438 // clang-format on
05439
05440 struct XmlReporter : public IReporter
05441 {
05442     XmlWriter xml;
05443     DOCTEST_DECLARE_MUTEX(mutex)
05444
05445     // caching pointers/references to objects of these types - safe to do
05446     const ContextOptions& opt;
05447     const TestCaseData* tc = nullptr;
05448
05449     XmlReporter(const ContextOptions& co)
05450         : xml(*co.cout)
05451         , opt(co) {}
05452
05453     void log_contexts() {
05454         int num_contexts = get_num_active_contexts();
05455         if(num_contexts) {
05456             auto contexts = get_active_contexts();
05457             std::stringstream ss;
05458             for(int i = 0; i < num_contexts; ++i) {
05459                 contexts[i]->stringify(&ss);
05460                 xml.scopedElement("Info").writeText(ss.str());
05461                 ss.str("");
05462             }
05463         }
05464     }
05465
05466     unsigned line(unsigned l) const { return opt.no_line_numbers ? 0 : l; }
05467
05468     void test_case_start_impl(const TestCaseData& in) {
05469         bool open_ts_tag = false;
05470         if(tc != nullptr) { // we have already opened a test suite
05471             if(std::strcmp(tc->m_test_suite, in.m_test_suite) != 0) {
05472                 xml.endElement();
05473                 open_ts_tag = true;
05474             }
05475         }
05476         else {
05477             open_ts_tag = true; // first test case ==> first test suite
05478         }
05479
05480         if(open_ts_tag) {
05481             xml.startElement("TestSuite");
05482             xml.writeAttribute("name", in.m_test_suite);
05483         }
05484
05485         tc = &in;
05486         xml.startElement("TestCase")
05487             .writeAttribute("name", in.m_name)
05488             .writeAttribute("filename", skipPathFromFilename(in.m_file.c_str()))
05489             .writeAttribute("line", line(in.m_line))
05490             .writeAttribute("description", in.m_description);
05491
05492         if(Approx(in.m_timeout) != 0)
05493             xml.writeAttribute("timeout", in.m_timeout);
05494         if(in.m_may_fail)
05495             xml.writeAttribute("may_fail", true);
05496         if(in.m_should_fail)
05497             xml.writeAttribute("should_fail", true);
05498     }
05499
05500     // =====
05501     // WHAT FOLLOWS ARE OVERRIDES OF THE VIRTUAL METHODS OF THE REPORTER INTERFACE
05502     // =====
05503
05504     void report_query(const QueryData& in) override {
05505         test_run_start();
05506         if(opt.list_reporters) {
05507             for(auto& curr : getListeners())
05508                 xml.scopedElement("Listener")
05509                     .writeAttribute("priority", curr.first.first)
05510                     .writeAttribute("name", curr.first.second);
05511             for(auto& curr : getReporters())
05512                 xml.scopedElement("Reporter")
05513                     .writeAttribute("priority", curr.first.first)
05514                     .writeAttribute("name", curr.first.second);
05515         } else if(opt.count || opt.list_test_cases) {
05516             for(unsigned i = 0; i < in.num_data; ++i) {
05517                 xml.scopedElement("TestCase").writeAttribute("name", in.data[i]->m_name)
05518                     .writeAttribute("testsuite", in.data[i]->m_test_suite)
05519                     .writeAttribute("filename", skipPathFromFilename(in.data[i]->m_file.c_str()))
05520                     .writeAttribute("line", line(in.data[i]->m_line))

```

```

05521         .writeAttribute("skipped", in.data[i]->m_skip);
05522     }
05523     xml.scopedElement("OverallResultsTestCases")
05524         .writeAttribute("unskipped", in.run_stats->numTestCasesPassingFilters);
05525 } else if(opt.list_test_suites) {
05526     for(unsigned i = 0; i < in.num_data; ++i)
05527         xml.scopedElement("TestSuite").writeAttribute("name", in.data[i]->m_test_suite);
05528     xml.scopedElement("OverallResultsTestCases")
05529         .writeAttribute("unskipped", in.run_stats->numTestCasesPassingFilters);
05530     xml.scopedElement("OverallResultsTestSuites")
05531         .writeAttribute("unskipped", in.run_stats->numTestSuitesPassingFilters);
05532 }
05533 xml.endElement();
05534 }
05535
05536 void test_run_start() override {
05537     xml.writeDeclaration();
05538
05539     // remove .exe extension - mainly to have the same output on UNIX and Windows
05540     std::string binary_name = skipPathFromFilename(opt.binary_name.c_str());
05541 #ifdef DOCTEST_PLATFORM_WINDOWS
05542     if(binary_name.rfind(".exe") != std::string::npos)
05543         binary_name = binary_name.substr(0, binary_name.length() - 4);
05544 #endif // DOCTEST_PLATFORM_WINDOWS
05545
05546     xml.startElement("doctest").writeAttribute("binary", binary_name);
05547     if(opt.no_version == false)
05548         xml.writeAttribute("version", DOCTEST_VERSION_STR);
05549
05550     // only the consequential ones (TODO: filters)
05551     xml.scopedElement("Options")
05552         .writeAttribute("order_by", opt.order_by.c_str())
05553         .writeAttribute("rand_seed", opt.rand_seed)
05554         .writeAttribute("first", opt.first)
05555         .writeAttribute("last", opt.last)
05556         .writeAttribute("abort_after", opt.abort_after)
05557         .writeAttribute("subcase_filter_levels", opt.subcase_filter_levels)
05558         .writeAttribute("case_sensitive", opt.case_sensitive)
05559         .writeAttribute("no_throw", opt.no_throw)
05560         .writeAttribute("no_skip", opt.no_skip);
05561 }
05562
05563 void test_run_end(const TestRunStats& p) override {
05564     if(tc) // the TestSuite tag - only if there has been at least 1 test case
05565         xml.endElement();
05566
05567     xml.scopedElement("OverallResultsAsserts")
05568         .writeAttribute("successes", p.numAsserts - p.numAssertsFailed)
05569         .writeAttribute("failures", p.numAssertsFailed);
05570
05571     xml.startElement("OverallResultsTestCases")
05572         .writeAttribute("successes",
05573             p.numTestCasesPassingFilters - p.numTestCasesFailed)
05574         .writeAttribute("failures", p.numTestCasesFailed);
05575     if(opt.no_skipped_summary == false)
05576         xml.writeAttribute("skipped", p.numTestCases - p.numTestCasesPassingFilters);
05577     xml.endElement();
05578
05579     xml.endElement();
05580 }
05581
05582 void test_case_start(const TestCaseData& in) override {
05583     test_case_start_impl(in);
05584     xml.ensureTagClosed();
05585 }
05586
05587 void test_case_reenter(const TestCaseData&) override {}
05588
05589 void test_case_end(const CurrentTestCaseStats& st) override {
05590     xml.startElement("OverallResultsAsserts")
05591         .writeAttribute("successes",
05592             st.numAssertsCurrentTest - st.numAssertsFailedCurrentTest)
05593         .writeAttribute("failures", st.numAssertsFailedCurrentTest)
05594         .writeAttribute("test_case_success", st.testCaseSuccess);
05595     if(opt.duration)
05596         xml.writeAttribute("duration", st.seconds);
05597     if(tc->m_expected_failures)
05598         xml.writeAttribute("expected_failures", tc->m_expected_failures);
05599     xml.endElement();
05600
05601     xml.endElement();
05602 }
05603
05604 void test_case_exception(const TestCaseException& e) override {
05605     DOCTEST_LOCK_MUTEX(mutex)
05606
05607     xml.scopedElement("Exception")

```

```

05608         .writeAttribute("crash", e.is_crash)
05609         .writeText(e.error_string.c_str());
05610     }
05611
05612     void subcase_start(const SubcaseSignature& in) override {
05613         xml.startElement("SubCase")
05614             .writeAttribute("name", in.m_name)
05615             .writeAttribute("filename", skipPathFromFilename(in.m_file))
05616             .writeAttribute("line", line(in.m_line));
05617         xml.ensureTagClosed();
05618     }
05619
05620     void subcase_end() override { xml.endElement(); }
05621
05622     void log_assert(const AssertData& rb) override {
05623         if(!rb.m_failed && !opt.success)
05624             return;
05625
05626         DOCTEST_LOCK_MUTEX(mutex)
05627
05628         xml.startElement("Expression")
05629             .writeAttribute("success", !rb.m_failed)
05630             .writeAttribute("type", assertString(rb.m_at))
05631             .writeAttribute("filename", skipPathFromFilename(rb.m_file))
05632             .writeAttribute("line", line(rb.m_line));
05633
05634         xml.scopedElement("Original").writeText(rb.m_expr);
05635
05636         if(rb.m_threw)
05637             xml.scopedElement("Exception").writeText(rb.m_exception.c_str());
05638
05639         if(rb.m_at & assertType::is_throws_as)
05640             xml.scopedElement("ExpectedException").writeText(rb.m_exception_type);
05641         if(rb.m_at & assertType::is_throws_with)
05642             xml.scopedElement("ExpectedExceptionString").writeText(rb.m_exception_string.c_str());
05643         if((rb.m_at & assertType::is_normal) && !rb.m_threw)
05644             xml.scopedElement("Expanded").writeText(rb.m_decomp.c_str());
05645
05646         log_contexts();
05647
05648         xml.endElement();
05649     }
05650
05651     void log_message(const MessageData& mb) override {
05652         DOCTEST_LOCK_MUTEX(mutex)
05653
05654         xml.startElement("Message")
05655             .writeAttribute("type", failureString(mb.m_severity))
05656             .writeAttribute("filename", skipPathFromFilename(mb.m_file))
05657             .writeAttribute("line", line(mb.m_line));
05658
05659         xml.scopedElement("Text").writeText(mb.m_string.c_str());
05660
05661         log_contexts();
05662
05663         xml.endElement();
05664     }
05665
05666     void test_case_skipped(const TestCaseData& in) override {
05667         if(opt.no_skipped_summary == false) {
05668             test_case_start_impl(in);
05669             xml.writeAttribute("skipped", "true");
05670             xml.endElement();
05671         }
05672     }
05673 };
05674
05675 DOCTEST_REGISTER_REPORTER("xml", 0, XmlReporter);
05676
05677 void fulltext_log_assert_to_stream(std::ostream& s, const AssertData& rb) {
05678     if((rb.m_at & (assertType::is_throws_as | assertType::is_throws_with)) ==
05679        0)
05680         s << Color::Cyan << assertString(rb.m_at) << "( " << rb.m_expr << " ) "
05681         << Color::None;
05682
05683     if(rb.m_at & assertType::is_throws) {
05684         s << (rb.m_threw ? "threw as expected!" : "did NOT throw at all!") << "\n";
05685     } else if((rb.m_at & assertType::is_throws_as) &&
05686              (rb.m_at & assertType::is_throws_with)) {
05687         s << Color::Cyan << assertString(rb.m_at) << "( " << rb.m_expr << ", \""
05688         << rb.m_exception_string.c_str()
05689         << "\", " << rb.m_exception_type << " ) " << Color::None;
05690         if(rb.m_threw) {
05691             if(!rb.m_failed) {
05692                 s << "threw as expected!\n";
05693             } else {
05694                 s << "threw a DIFFERENT exception! (contents: " << rb.m_exception << ")\n";

```

```

05695     }
05696     } else {
05697         s « "did NOT throw at all!\n";
05698     }
05699 } else if(rb.m_at &
05700     assertType::is_throws_as) {
05701     s « Color::Cyan « assertString(rb.m_at) « "( " « rb.m_expr « ", "
05702     « rb.m_exception_type « " ) " « Color::None
05703     « (rb.m_threw ? (rb.m_threw_as ? "threw as expected!" :
05704     "threw a DIFFERENT exception: ") :
05705     "did NOT throw at all!")
05706     « Color::Cyan « rb.m_exception « "\n";
05707 } else if(rb.m_at &
05708     assertType::is_throws_with) {
05709     s « Color::Cyan « assertString(rb.m_at) « "( " « rb.m_expr « ", \"\"
05710     « rb.m_exception_string.c_str()
05711     « "\" ) " « Color::None
05712     « (rb.m_threw ? (!rb.m_failed ? "threw as expected!" :
05713     "threw a DIFFERENT exception: ") :
05714     "did NOT throw at all!")
05715     « Color::Cyan « rb.m_exception « "\n";
05716 } else if(rb.m_at & assertType::is_nothrow) {
05717     s « (rb.m_threw ? "THREW exception: " : "didn't throw!") « Color::Cyan
05718     « rb.m_exception « "\n";
05719 } else {
05720     s « (rb.m_threw ? "THREW exception: " :
05721     (!rb.m_failed ? "is correct!\n" : "is NOT correct!\n"));
05722     if(rb.m_threw)
05723         s « rb.m_exception « "\n";
05724     else
05725         s « " values: " « assertString(rb.m_at) « "( " « rb.m_decomp « " )\n";
05726 }
05727 }
05728
05729 // TODO:
05730 // - log_message()
05731 // - respond to queries
05732 // - honor remaining options
05733 // - more attributes in tags
05734 struct JUnitReporter : public IReporter
05735 {
05736     XmlWriter xml;
05737     DOCTEST_DECLARE_MUTEX(mutex)
05738     Timer timer;
05739     std::vector<String> deepestSubcaseStackNames;
05740
05741     struct JUnitTestCaseData
05742     {
05743         static std::string getTimestamp() {
05744             // Beware, this is not reentrant because of backward compatibility issues
05745             // Also, UTC only, again because of backward compatibility (%Z is C++11)
05746             time_t rawtime;
05747             std::time(&rawtime);
05748             auto const timeStampSize = sizeof("2017-01-16T17:06:45Z");
05749
05750             std::tm timeInfo;
05751 #ifdef DOCTEST_PLATFORM_WINDOWS
05752             gmtime_s(&timeInfo, &rawtime);
05753 #else // DOCTEST_PLATFORM_WINDOWS
05754             gmtime_r(&rawtime, &timeInfo);
05755 #endif // DOCTEST_PLATFORM_WINDOWS
05756
05757             char timeStamp[timeStampSize];
05758             const char* const fmt = "%Y-%m-%dT%H:%M:%SZ";
05759
05760             std::strftime(timeStamp, timeStampSize, fmt, &timeInfo);
05761             return std::string(timeStamp);
05762         }
05763
05764         struct JUnitTestMessage
05765         {
05766             JUnitTestMessage(const std::string& _message, const std::string& _type, const
std::string& _details)
05767                 : message(_message), type(_type), details(_details) {}
05768
05769             JUnitTestMessage(const std::string& _message, const std::string& _details)
05770                 : message(_message), type(), details(_details) {}
05771
05772             std::string message, type, details;
05773         };
05774
05775         struct JUnitTestCase
05776         {
05777             JUnitTestCase(const std::string& _classname, const std::string& _name)
05778                 : classname(_classname), name(_name), time(0), failures() {}
05779
05780             std::string classname, name;

```

```

05781         double time;
05782         std::vector<JUnitTestMessage> failures, errors;
05783     };
05784
05785     void add(const std::string& classname, const std::string& name) {
05786         testcases.emplace_back(classname, name);
05787     }
05788
05789     void appendSubcaseNamesToLastTestcase(std::vector<String> nameStack) {
05790         for(auto& curr: nameStack)
05791             if(curr.size())
05792                 testcases.back().name += std::string("/") + curr.c_str();
05793     }
05794
05795     void addTime(double time) {
05796         if(time < 1e-4)
05797             time = 0;
05798         testcases.back().time = time;
05799         totalSeconds += time;
05800     }
05801
05802     void addFailure(const std::string& message, const std::string& type, const std::string&
details) {
05803         testcases.back().failures.emplace_back(message, type, details);
05804         ++totalFailures;
05805     }
05806
05807     void addError(const std::string& message, const std::string& details) {
05808         testcases.back().errors.emplace_back(message, details);
05809         ++totalErrors;
05810     }
05811
05812     std::vector<JUnitTestCase> testcases;
05813     double totalSeconds = 0;
05814     int totalErrors = 0, totalFailures = 0;
05815 };
05816
05817 JUnitTestCaseData testCaseData;
05818
05819 // caching pointers/references to objects of these types - safe to do
05820 const ContextOptions& opt;
05821 const TestCaseData* tc = nullptr;
05822
05823 JUnitReporter(const ContextOptions& co)
05824     : xml(*co.cout)
05825     , opt(co) {}
05826
05827 unsigned line(unsigned l) const { return opt.no_line_numbers ? 0 : l; }
05828
05829 // =====
05830 // WHAT FOLLOWS ARE OVERRIDES OF THE VIRTUAL METHODS OF THE REPORTER INTERFACE
05831 // =====
05832
05833 void report_query(const QueryData&) override {
05834     xml.writeDeclaration();
05835 }
05836
05837 void test_run_start() override {
05838     xml.writeDeclaration();
05839 }
05840
05841 void test_run_end(const TestRunStats& p) override {
05842     // remove .exe extension - mainly to have the same output on UNIX and Windows
05843     std::string binary_name = skipPathFromFilename(opt.binary_name.c_str());
05844 #ifdef DOCTEST_PLATFORM_WINDOWS
05845     if(binary_name.rfind(".exe") != std::string::npos)
05846         binary_name = binary_name.substr(0, binary_name.length() - 4);
05847 #endif // DOCTEST_PLATFORM_WINDOWS
05848     xml.startElement("testsuites");
05849     xml.startElement("testsuite").writeAttribute("name", binary_name)
05850         .writeAttribute("errors", testCaseData.totalErrors)
05851         .writeAttribute("failures", testCaseData.totalFailures)
05852         .writeAttribute("tests", p.numAsserts);
05853     if(opt.no_time_in_output == false) {
05854         xml.writeAttribute("time", testCaseData.totalSeconds);
05855         xml.writeAttribute("timestamp", JUnitTestCaseData::getCurrentTimestamp());
05856     }
05857     if(opt.no_version == false)
05858         xml.writeAttribute("doctest_version", DOCTEST_VERSION_STR);
05859
05860     for(const auto& testCase : testCaseData.testcases) {
05861         xml.startElement("testcase")
05862             .writeAttribute("classname", testCase.classname)
05863             .writeAttribute("name", testCase.name);
05864         if(opt.no_time_in_output == false)
05865             xml.writeAttribute("time", testCase.time);
05866         // This is not ideal, but it should be enough to mimic gtest's junit output.

```

```

05867         xml.writeAttribute("status", "run");
05868
05869         for(const auto& failure : testCase.failures) {
05870             xml.scopedElement("failure")
05871                 .writeAttribute("message", failure.message)
05872                 .writeAttribute("type", failure.type)
05873                 .writeText(failure.details, false);
05874         }
05875
05876         for(const auto& error : testCase.errors) {
05877             xml.scopedElement("error")
05878                 .writeAttribute("message", error.message)
05879                 .writeText(error.details);
05880         }
05881
05882         xml.endElement();
05883     }
05884     xml.endElement();
05885     xml.endElement();
05886 }
05887
05888 void test_case_start(const TestCaseData& in) override {
05889     testCaseData.add(skipPathFromFilename(in.m_file.c_str()), in.m_name);
05890     timer.start();
05891 }
05892
05893 void test_case_reenter(const TestCaseData& in) override {
05894     testCaseData.addTime(timer.getElapsedSeconds());
05895     testCaseData.appendSubcaseNamesToLastTestcase(deepestSubcaseStackNames);
05896     deepestSubcaseStackNames.clear();
05897
05898     timer.start();
05899     testCaseData.add(skipPathFromFilename(in.m_file.c_str()), in.m_name);
05900 }
05901
05902 void test_case_end(const CurrentTestCaseStats&) override {
05903     testCaseData.addTime(timer.getElapsedSeconds());
05904     testCaseData.appendSubcaseNamesToLastTestcase(deepestSubcaseStackNames);
05905     deepestSubcaseStackNames.clear();
05906 }
05907
05908 void test_case_exception(const TestCaseException& e) override {
05909     DOCTEST_LOCK_MUTEX(mutex)
05910     testCaseData.addError("exception", e.error_string.c_str());
05911 }
05912
05913 void subcase_start(const SubcaseSignature& in) override {
05914     deepestSubcaseStackNames.push_back(in.m_name);
05915 }
05916
05917 void subcase_end() override {}
05918
05919 void log_assert(const AssertData& rb) override {
05920     if(!rb.m_failed) // report only failures & ignore the `success` option
05921         return;
05922
05923     DOCTEST_LOCK_MUTEX(mutex)
05924
05925     std::ostringstream os;
05926     os << skipPathFromFilename(rb.m_file) << (opt.gnu_file_line ? ":" : "(")
05927         << line(rb.m_line) << (opt.gnu_file_line ? ":" : ")") << std::endl;
05928
05929     fulltext_log_assert_to_stream(os, rb);
05930     log_contexts(os);
05931     testCaseData.addFailure(rb.m_decomp.c_str(), assertString(rb.m_at), os.str());
05932 }
05933
05934 void log_message(const MessageData& mb) override {
05935     if(mb.m_severity & assertType::is_warn) // report only failures
05936         return;
05937
05938     DOCTEST_LOCK_MUTEX(mutex)
05939
05940     std::ostringstream os;
05941     os << skipPathFromFilename(mb.m_file) << (opt.gnu_file_line ? ":" : "(")
05942         << line(mb.m_line) << (opt.gnu_file_line ? ":" : ")") << std::endl;
05943
05944     os << mb.m_string.c_str() << "\n";
05945     log_contexts(os);
05946
05947     testCaseData.addFailure(mb.m_string.c_str(),
05948         mb.m_severity & assertType::is_check ? "FAIL_CHECK" : "FAIL", os.str());
05949 }
05950
05951 void test_case_skipped(const TestCaseData&) override {}
05952
05953 void log_contexts(std::ostringstream& s) {

```



```

05954         int num_contexts = get_num_active_contexts();
05955         if(num_contexts) {
05956             auto contexts = get_active_contexts();
05957
05958             s « " logged: ";
05959             for(int i = 0; i < num_contexts; ++i) {
05960                 s « (i == 0 ? "" : " ");
05961                 contexts[i]->stringify(&s);
05962                 s « std::endl;
05963             }
05964         }
05965     }
05966 };
05967
05968 DOCTEST_REGISTER_REPORTER("junit", 0, JUnitReporter);
05969
05970 struct Whitespace
05971 {
05972     int nrSpaces;
05973     explicit Whitespace(int nr)
05974         : nrSpaces(nr) {}
05975 };
05976
05977 std::ostream& operator<<(std::ostream& out, const Whitespace& ws) {
05978     if(ws.nrSpaces != 0)
05979         out « std::setw(ws.nrSpaces) « ' ';
05980     return out;
05981 }
05982
05983 struct ConsoleReporter : public IReporter
05984 {
05985     std::ostream& s;
05986     bool hasLoggedCurrentTestStart;
05987     std::vector<SubcaseSignature> subcasesStack;
05988     size_t currentSubcaseLevel;
05989     DOCTEST_DECLARE_MUTEX(mutex)
05990
05991     // caching pointers/references to objects of these types - safe to do
05992     const ContextOptions& opt;
05993     const TestCaseData* tc;
05994
05995     ConsoleReporter(const ContextOptions& co)
05996         : s(*co.cout)
05997         , opt(co) {}
05998
05999     ConsoleReporter(const ContextOptions& co, std::ostream& ostr)
06000         : s(ostr)
06001         , opt(co) {}
06002
06003     // =====
06004     // WHAT FOLLOWS ARE HELPERS USED BY THE OVERRIDES OF THE VIRTUAL METHODS OF THE INTERFACE
06005     // =====
06006
06007     void separator_to_stream() {
06008         s « Color::Yellow
06009         « "=====
06010         « "\n";
06011     }
06012
06013     const char* getSuccessOrFailString(bool success, assertType::Enum at,
06014         const char* success_str) {
06015         if(success)
06016             return success_str;
06017         return failureString(at);
06018     }
06019
06020     Color::Enum getSuccessOrFailColor(bool success, assertType::Enum at) {
06021         return success ? Color::BrightGreen :
06022             (at & assertType::is_warn) ? Color::Yellow : Color::Red;
06023     }
06024
06025     void successOrFailColoredStringToStream(bool success, assertType::Enum at,
06026         const char* success_str = "SUCCESS") {
06027         s « getSuccessOrFailColor(success, at)
06028         « getSuccessOrFailString(success, at, success_str) « ": ";
06029     }
06030
06031     void log_contexts() {
06032         int num_contexts = get_num_active_contexts();
06033         if(num_contexts) {
06034             auto contexts = get_active_contexts();
06035
06036             s « Color::None « " logged: ";
06037             for(int i = 0; i < num_contexts; ++i) {
06038                 s « (i == 0 ? "" : " ");
06039                 contexts[i]->stringify(&s);
06040                 s « "\n";

```

```

06041         }
06042     }
06043
06044     s << "\n";
06045 }
06046
06047 // this was requested to be made virtual so users could override it
06048 virtual void file_line_to_stream(const char* file, int line,
06049                                const char* tail = "") {
06050     s << Color::LightGrey << skipPathFromFilename(file) << (opt.gnu_file_line ? ":" : "(")
06051     << (opt.no_line_numbers ? 0 : line) // 0 or the real num depending on the option
06052     << (opt.gnu_file_line ? ":" : ")") << tail;
06053 }
06054
06055 void logTestStart() {
06056     if(hasLoggedCurrentTestStart)
06057         return;
06058
06059     separator_to_stream();
06060     file_line_to_stream(tc->m_file.c_str(), tc->m_line, "\n");
06061     if(tc->m_description)
06062         s << Color::Yellow << "DESCRIPTION: " << Color::None << tc->m_description << "\n";
06063     if(tc->m_test_suite && tc->m_test_suite[0] != '\0')
06064         s << Color::Yellow << "TEST SUITE: " << Color::None << tc->m_test_suite << "\n";
06065     if(strncmp(tc->m_name, " Scenario:", 11) != 0)
06066         s << Color::Yellow << "TEST CASE: ";
06067     s << Color::None << tc->m_name << "\n";
06068
06069     for(size_t i = 0; i < currentSubcaseLevel; ++i) {
06070         if(subcasesStack[i].m_name[0] != '\0')
06071             s << " " << subcasesStack[i].m_name << "\n";
06072     }
06073
06074     if(currentSubcaseLevel != subcasesStack.size()) {
06075         s << Color::Yellow << "\nDEEPEST SUBCASE STACK REACHED (DIFFERENT FROM THE CURRENT
06076 ONE):\n" << Color::None;
06077         for(size_t i = 0; i < subcasesStack.size(); ++i) {
06078             if(subcasesStack[i].m_name[0] != '\0')
06079                 s << " " << subcasesStack[i].m_name << "\n";
06080         }
06081
06082         s << "\n";
06083
06084         hasLoggedCurrentTestStart = true;
06085     }
06086
06087 void printVersion() {
06088     if(opt.no_version == false)
06089         s << Color::Cyan << "[doctest] " << Color::None << "doctest version is \""
06090         << DOCTEST_VERSION_STR << "\"\n";
06091 }
06092
06093 void printIntro() {
06094     if(opt.no_intro == false) {
06095         printVersion();
06096         s << Color::Cyan << "[doctest] " << Color::None
06097         << "run with \"--\" DOCTEST_OPTIONS_PREFIX_DISPLAY \"help\" for options\n";
06098     }
06099 }
06100
06101 void printHelp() {
06102     int sizePrefixDisplay = static_cast<int>(strlen(DOCTEST_OPTIONS_PREFIX_DISPLAY));
06103     printVersion();
06104     // clang-format off
06105     s << Color::Cyan << "[doctest]\n" << Color::None;
06106     s << Color::Cyan << "[doctest] " << Color::None;
06107     s << "boolean values: \"1/on/yes/true\" or \"0/off/no/false\"\n";
06108     s << Color::Cyan << "[doctest] " << Color::None;
06109     s << "filter values: \"str1,str2,str3\" (comma separated strings)\n";
06110     s << Color::Cyan << "[doctest]\n" << Color::None;
06111     s << Color::Cyan << "[doctest] " << Color::None;
06112     s << "filters use wildcards for matching strings\n";
06113     s << Color::Cyan << "[doctest] " << Color::None;
06114     s << "something passes a filter if any of the strings in a filter matches\n";
06115 #ifndef DOCTEST_CONFIG_NO_UNPREFIXED_OPTIONS
06116     s << Color::Cyan << "[doctest]\n" << Color::None;
06117     s << Color::Cyan << "[doctest] " << Color::None;
06118     s << "ALL FLAGS, OPTIONS AND FILTERS ALSO AVAILABLE WITH A \""
06119     DOCTEST_CONFIG_OPTIONS_PREFIX " \" PREFIX!!!\n";
06120 #endif
06121     s << Color::Cyan << "[doctest]\n" << Color::None;
06122     s << Color::Cyan << "[doctest] " << Color::None;
06123     s << "Query flags - the program quits after them. Available:\n";
06124     s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "?", --" DOCTEST_OPTIONS_PREFIX_DISPLAY "help, -"
    DOCTEST_OPTIONS_PREFIX_DISPLAY "h
    << Whitespace(sizePrefixDisplay*0) << "prints this message\n";

```

```

06125         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "v, --" DOCTEST_OPTIONS_PREFIX_DISPLAY "version
06126         << Whitespace(sizePrefixDisplay*1) << "prints the version\n";
06127         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "c, --" DOCTEST_OPTIONS_PREFIX_DISPLAY "count
06128         << Whitespace(sizePrefixDisplay*1) << "prints the number of matching tests\n";
06129         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "ltc, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06130         "list-test-cases" << Whitespace(sizePrefixDisplay*1) << "lists all matching tests by name\n";
06131         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "lts, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06132         "list-test-suites" << Whitespace(sizePrefixDisplay*1) << "lists all matching test suites\n";
06133         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "lr, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06134         "list-reporters" << Whitespace(sizePrefixDisplay*1) << "lists all registered reporters\n\n";
06135         // ===== << 79
06136         s << Color::Cyan << "[doctest] " << Color::None;
06137         s << "The available <int>/<string> options/filters are:\n\n";
06138         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "tc, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06139         "test-case=<filters>" << Whitespace(sizePrefixDisplay*1) << "filters tests by their name\n";
06140         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "tce, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06141         "test-case-exclude=<filters>" << Whitespace(sizePrefixDisplay*1) << "filters OUT tests by their name\n";
06142         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "sf, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06143         "source-file=<filters>" << Whitespace(sizePrefixDisplay*1) << "filters tests by their file\n";
06144         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "sfe, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06145         "source-file-exclude=<filters>" << Whitespace(sizePrefixDisplay*1) << "filters OUT tests by their file\n";
06146         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "ts, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06147         "test-suite=<filters>" << Whitespace(sizePrefixDisplay*1) << "filters tests by their test suite\n";
06148         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "tse, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06149         "test-suite-exclude=<filters>" << Whitespace(sizePrefixDisplay*1) << "filters OUT tests by their test suite\n";
06150         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "sc, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06151         "subcase=<filters>" << Whitespace(sizePrefixDisplay*1) << "filters subcases by their name\n";
06152         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "sce, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06153         "subcase-exclude=<filters>" << Whitespace(sizePrefixDisplay*1) << "filters OUT subcases by their name\n";
06154         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "r, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06155         "reporters=<filters>" << Whitespace(sizePrefixDisplay*1) << "reporters to use (console is default)\n";
06156         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "o, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06157         "out=<string>" << Whitespace(sizePrefixDisplay*1) << "output filename\n";
06158         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "ob, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06159         "order-by=<string>" << Whitespace(sizePrefixDisplay*1) << "how the tests should be ordered\n";
06160         s << Whitespace(sizePrefixDisplay*3) << " <string> -
06161         [file/suite/name/rand/none]\n";
06161         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "rs, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06162         "rand-seed=<int>" << Whitespace(sizePrefixDisplay*1) << "seed for random ordering\n";
06163         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "f, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06164         "first=<int>" << Whitespace(sizePrefixDisplay*1) << "the first test passing the filters to\n";
06165         s << Whitespace(sizePrefixDisplay*3) << " execute -
06166         for range-based execution\n";
06166         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "l, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06167         "last=<int>" << Whitespace(sizePrefixDisplay*1) << "the last test passing the filters to\n";
06168         s << Whitespace(sizePrefixDisplay*3) << " execute -
06169         for range-based execution\n";
06169         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "aa, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06170         "abort-after=<int>" << Whitespace(sizePrefixDisplay*1) << "stop after <int> failed assertions\n";
06171         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "scfl, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06172         "subcase-filter-levels=<int>" << Whitespace(sizePrefixDisplay*1) << "apply filters for the first <int> levels\n";
06173         s << Color::Cyan << "\n[doctest] " << Color::None;
06174         s << "Bool options - can be used like flags and true is assumed. Available:\n\n";
06175         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "s, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06176         "success=<bool>" << Whitespace(sizePrefixDisplay*1) << "include successful assertions in output\n";
06177         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "cs, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06178         "case-sensitive=<bool>" << Whitespace(sizePrefixDisplay*1) << "filters being treated as case sensitive\n";
06179         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "e, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06180         "exit=<bool>" << Whitespace(sizePrefixDisplay*1) << "exits after the tests finish\n";
06181         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "d, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
06182         "duration=<bool>" << Whitespace(sizePrefixDisplay*1) << "prints the time duration of each test\n";
06183         s << " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "m, --" DOCTEST_OPTIONS_PREFIX_DISPLAY

```

```

    "minimal=<bool>                "
06184    « Whitespace(sizePrefixDisplay*1) « "minimal console output (only failures)\n";
06185    s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "q, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"quiet=<bool>                "
06186    « Whitespace(sizePrefixDisplay*1) « "no console output\n";
06187    s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "nt, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-throw=<bool>            "
06188    « Whitespace(sizePrefixDisplay*1) « "skips exceptions-related assert checks\n";
06189    s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "ne, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-exitcode=<bool>        "
06190    « Whitespace(sizePrefixDisplay*1) « "returns (or exits) always with success\n";
06191    s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "nr, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-run=<bool>              "
06192    « Whitespace(sizePrefixDisplay*1) « "skips all runtime doctest operations\n";
06193    s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "ni, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-intro=<bool>           "
06194    « Whitespace(sizePrefixDisplay*1) « "omit the framework intro in the output\n";
06195    s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "nv, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-version=<bool>         "
06196    « Whitespace(sizePrefixDisplay*1) « "omit the framework version in the output\n";
06197    s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "nc, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-colors=<bool>          "
06198    « Whitespace(sizePrefixDisplay*1) « "disables colors in output\n";
06199    s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "fc, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"force-colors=<bool>       "
06200    « Whitespace(sizePrefixDisplay*1) « "use colors even when not in a tty\n";
06201    s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "nb, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-breaks=<bool>          "
06202    « Whitespace(sizePrefixDisplay*1) « "disables breakpoints in debuggers\n";
06203    s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "ns, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-skip=<bool>            "
06204    « Whitespace(sizePrefixDisplay*1) « "don't skip test cases marked as skip\n";
06205    s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "gfl, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"gnu-file-line=<bool>      "
06206    « Whitespace(sizePrefixDisplay*1) « ":n: vs (n): for line numbers in output\n";
06207    s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "npf, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-path-filenames=<bool>  "
06208    « Whitespace(sizePrefixDisplay*1) « "only filenames and no paths in output\n";
06209    s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "spp, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"skip-path-prefixes=<p1:p2> "
06210    « Whitespace(sizePrefixDisplay*1) « "whenever file paths start with this prefix, remove
it from the output\n";
06211    s « " -" DOCTEST_OPTIONS_PREFIX_DISPLAY "nln, --" DOCTEST_OPTIONS_PREFIX_DISPLAY
"no-line-numbers=<bool>    "
06212    « Whitespace(sizePrefixDisplay*1) « "0 instead of real line numbers in output\n";
06213    // ===== « 79
06214    // clang-format on
06215
06216    s « Color::Cyan « "\n[doctest] " « Color::None;
06217    s « "for more information visit the project documentation\n\n";
06218    }
06219
06220    void printRegisteredReporters() {
06221        printVersion();
06222        auto printReporters = [this] (const reporterMap& reporters, const char* type) {
06223            if(reporters.size()) {
06224                s « Color::Cyan « "[doctest] " « Color::None « "listing all registered " « type «
"\n";
06225                for(auto& curr : reporters)
06226                    s « "priority: " « std::setw(5) « curr.first.first
« " name: " « curr.first.second « "\n";
06227            }
06228        };
06229        printReporters(getListeners(), "listeners");
06230        printReporters(getReporters(), "reporters");
06231    }
06232
06233    // =====
06234    // WHAT FOLLOWS ARE OVERRIDES OF THE VIRTUAL METHODS OF THE REPORTER INTERFACE
06235    // =====
06236
06237    void report_query(const QueryData& in) override {
06238        if(opt.version) {
06239            printVersion();
06240        } else if(opt.help) {
06241            printHelp();
06242        } else if(opt.list_reporters) {
06243            printRegisteredReporters();
06244        } else if(opt.count || opt.list_test_cases) {
06245            if(opt.list_test_cases) {
06246                s « Color::Cyan « "[doctest] " « Color::None
« "listing all test case names\n";
06247                separator_to_stream();
06248            }
06249
06250            for(unsigned i = 0; i < in.num_data; ++i)
06251                s « Color::None « in.data[i]->m_name « "\n";
06252
06253

```

```

06254
06255         separator_to_stream();
06256
06257         s « Color::Cyan « "[doctest] " « Color::None
06258         « "unskipped test cases passing the current filters: "
06259         « g_cs->numTestCasesPassingFilters « "\n";
06260
06261     } else if(opt.list_test_suites) {
06262         s « Color::Cyan « "[doctest] " « Color::None « "listing all test suites\n";
06263         separator_to_stream();
06264
06265         for(unsigned i = 0; i < in.num_data; ++i)
06266             s « Color::None « in.data[i]->m_test_suite « "\n";
06267
06268         separator_to_stream();
06269
06270         s « Color::Cyan « "[doctest] " « Color::None
06271         « "unskipped test cases passing the current filters: "
06272         « g_cs->numTestCasesPassingFilters « "\n";
06273         s « Color::Cyan « "[doctest] " « Color::None
06274         « "test suites with unskipped test cases passing the current filters: "
06275         « g_cs->numTestSuitesPassingFilters « "\n";
06276     }
06277 }
06278
06279 void test_run_start() override {
06280     if(!opt.minimal)
06281         printIntro();
06282 }
06283
06284 void test_run_end(const TestRunStats& p) override {
06285     if(opt.minimal && p.numTestCasesFailed == 0)
06286         return;
06287
06288     separator_to_stream();
06289     s « std::dec;
06290
06291     auto totwidth =
06292     int(std::ceil(log10(static_cast<double>(std::max(p.numTestCasesPassingFilters,
06293     static_cast<unsigned>(p.numAsserts))) + 1)));
06294     auto passwidth =
06295     int(std::ceil(log10(static_cast<double>(std::max(p.numTestCasesPassingFilters - p.numTestCasesFailed,
06296     static_cast<unsigned>(p.numAsserts - p.numAssertsFailed))) + 1)));
06297     auto failwidth = int(std::ceil(log10(static_cast<double>(std::max(p.numTestCasesFailed,
06298     static_cast<unsigned>(p.numAssertsFailed))) + 1)));
06299     const bool anythingFailed = p.numTestCasesFailed > 0 || p.numAssertsFailed > 0;
06300     s « Color::Cyan « "[doctest] " « Color::None « "test cases: " « std::setw(totwidth)
06301     « p.numTestCasesPassingFilters « " | "
06302     « ((p.numTestCasesPassingFilters == 0 || anythingFailed) ? Color::None :
06303     Color::Green)
06304     « std::setw(passwidth) « p.numTestCasesPassingFilters - p.numTestCasesFailed « " passed"
06305     « Color::None « " | " « (p.numTestCasesFailed > 0 ? Color::Red : Color::None)
06306     « std::setw(failwidth) « p.numTestCasesFailed « " failed" « Color::None « " |";
06307     if(opt.no_skipped_summary == false) {
06308         const int numSkipped = p.numTestCases - p.numTestCasesPassingFilters;
06309         s « " " « (numSkipped == 0 ? Color::None : Color::Yellow) « numSkipped
06310         « " skipped" « Color::None;
06311     }
06312     s « "\n";
06313     s « Color::Cyan « "[doctest] " « Color::None « "assertions: " « std::setw(totwidth)
06314     « p.numAsserts « " | "
06315     « ((p.numAsserts == 0 || anythingFailed) ? Color::None : Color::Green)
06316     « std::setw(passwidth) « (p.numAsserts - p.numAssertsFailed) « " passed" « Color::None
06317     « " | " « (p.numAssertsFailed > 0 ? Color::Red : Color::None) « std::setw(failwidth)
06318     « p.numAssertsFailed « " failed" « Color::None « " | \n";
06319     s « Color::Cyan « "[doctest] " « Color::None
06320     « "Status: " « (p.numTestCasesFailed > 0 ? Color::Red : Color::Green)
06321     « ((p.numTestCasesFailed > 0) ? "FAILURE!" : "SUCCESS!") « Color::None « std::endl;
06322 }
06323
06324 void test_case_start(const TestCaseData& in) override {
06325     hasLoggedCurrentTestStart = false;
06326     tc = &in;
06327     subcasesStack.clear();
06328     currentSubcaseLevel = 0;
06329 }
06330
06331 void test_case_reenter(const TestCaseData&) override {
06332     subcasesStack.clear();
06333 }
06334
06335 void test_case_end(const CurrentTestCaseStats& st) override {
06336     if(tc->m_no_output)
06337         return;
06338
06339     // log the preamble of the test case only if there is something
06340     // else to print - something other than that an assert has failed

```

```

06336         if(opt.duration ||
06337            (st.failure_flags && st.failure_flags !=
static_cast<int>(TestCaseFailureReason::AssertFailure)))
06338             logTestStart();
06339
06340         if(opt.duration)
06341             s << Color::None << std::setprecision(6) << std::fixed << st.seconds
06342             << " s: " << tc->m_name << "\n";
06343
06344         if(st.failure_flags & TestCaseFailureReason::Timeout)
06345             s << Color::Red << "Test case exceeded time limit of " << std::setprecision(6)
06346             << std::fixed << tc->m_timeout << "!\n";
06347
06348         if(st.failure_flags & TestCaseFailureReason::ShouldHaveFailedButDidnt) {
06349             s << Color::Red << "Should have failed but didn't! Marking it as failed!\n";
06350         } else if(st.failure_flags & TestCaseFailureReason::ShouldHaveFailedAndDid) {
06351             s << Color::Yellow << "Failed as expected so marking it as not failed\n";
06352         } else if(st.failure_flags & TestCaseFailureReason::CouldHaveFailedAndDid) {
06353             s << Color::Yellow << "Allowed to fail so marking it as not failed\n";
06354         } else if(st.failure_flags & TestCaseFailureReason::DidntFailExactlyNumTimes) {
06355             s << Color::Red << "Didn't fail exactly " << tc->m_expected_failures
06356             << " times so marking it as failed!\n";
06357         } else if(st.failure_flags & TestCaseFailureReason::FailedExactlyNumTimes) {
06358             s << Color::Yellow << "Failed exactly " << tc->m_expected_failures
06359             << " times as expected so marking it as not failed!\n";
06360         }
06361         if(st.failure_flags & TestCaseFailureReason::TooManyFailedAsserts) {
06362             s << Color::Red << "Aborting - too many failed asserts!\n";
06363         }
06364         s << Color::None; // lgTM [cpp/useless-expression]
06365     }
06366
06367     void test_case_exception(const TestCaseException& e) override {
06368         DOCTEST_LOCK_MUTEX(mutex)
06369         if(tc->m_no_output)
06370             return;
06371
06372         logTestStart();
06373
06374         file_line_to_stream(tc->m_file.c_str(), tc->m_line, " ");
06375         successOrFailColoredStringToStream(false, e.is_crash ? assertType::is_require :
06376                                           assertType::is_check);
06377         s << Color::Red << (e.is_crash ? "test case CRASHED: " : "test case THREW exception: ")
06378         << Color::Cyan << e.error_string << "\n";
06379
06380         int num_stringified_contexts = get_num_stringified_contexts();
06381         if(num_stringified_contexts) {
06382             auto stringified_contexts = get_stringified_contexts();
06383             s << Color::None << " logged: ";
06384             for(int i = num_stringified_contexts; i > 0; --i) {
06385                 s << (i == num_stringified_contexts ? "" : " ")
06386                 << stringified_contexts[i - 1] << "\n";
06387             }
06388         }
06389         s << "\n" << Color::None;
06390     }
06391
06392     void subcase_start(const SubcaseSignature& subc) override {
06393         subcasesStack.push_back(subc);
06394         ++currentSubcaseLevel;
06395         hasLoggedCurrentTestStart = false;
06396     }
06397
06398     void subcase_end() override {
06399         --currentSubcaseLevel;
06400         hasLoggedCurrentTestStart = false;
06401     }
06402
06403     void log_assert(const AssertData& rb) override {
06404         if(!rb.m_failed && !opt.success) || tc->m_no_output)
06405             return;
06406
06407         DOCTEST_LOCK_MUTEX(mutex)
06408
06409         logTestStart();
06410
06411         file_line_to_stream(rb.m_file, rb.m_line, " ");
06412         successOrFailColoredStringToStream(!rb.m_failed, rb.m_at);
06413
06414         fulltext_log_assert_to_stream(s, rb);
06415
06416         log_contexts();
06417     }
06418
06419     void log_message(const MessageData& mb) override {
06420         if(tc->m_no_output)
06421             return;

```

```

06422
06423         DOCTEST_LOCK_MUTEX(mutex)
06424
06425         logTestStart();
06426
06427         file_line_to_stream(mb.m_file, mb.m_line, " ");
06428         s « getSuccessOrFailColor(false, mb.m_severity)
06429           « getSuccessOrFailString(mb.m_severity & assertType::is_warn, mb.m_severity,
06430                                   "MESSAGE") « ": ";
06431         s « Color::None « mb.m_string « "\n";
06432         log_contexts();
06433     }
06434
06435     void test_case_skipped(const TestCaseData& override) {}
06436 };
06437
06438     DOCTEST_REGISTER_REPORTER("console", 0, ConsoleReporter);
06439
06440 #ifndef DOCTEST_PLATFORM_WINDOWS
06441     struct DebugOutputWindowReporter : public ConsoleReporter
06442     {
06443         DOCTEST_THREAD_LOCAL static std::ostringstream oss;
06444
06445         DebugOutputWindowReporter(const ContextOptions& co)
06446             : ConsoleReporter(co, oss) {}
06447
06448 #define DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(func, type, arg)
06449         void func(type arg) override {
06450             bool with_col = g_no_colors;
06451             g_no_colors = false;
06452             ConsoleReporter::func(arg);
06453             if(oss.tellp() != std::streampos{}) {
06454                 DOCTEST_DEBUG_OUTPUT_STRING(oss.str().c_str());
06455                 oss.str("");
06456             }
06457             g_no_colors = with_col;
06458         }
06459
06460         DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(test_run_start, DOCTEST_EMPTY, DOCTEST_EMPTY)
06461         DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(test_run_end, const TestRunStats&, in)
06462         DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(test_case_start, const TestCaseData&, in)
06463         DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(test_case_reenter, const TestCaseData&, in)
06464         DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(test_case_end, const CurrentTestCaseStats&, in)
06465         DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(test_case_exception, const TestCaseException&, in)
06466         DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(subcase_start, const SubcaseSignature&, in)
06467         DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(subcase_end, DOCTEST_EMPTY, DOCTEST_EMPTY)
06468         DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(log_assert, const AssertData&, in)
06469         DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(log_message, const MessageData&, in)
06470         DOCTEST_DEBUG_OUTPUT_REPORTER_OVERRIDE(test_case_skipped, const TestCaseData&, in)
06471     };
06472
06473     DOCTEST_THREAD_LOCAL std::ostringstream DebugOutputWindowReporter::oss;
06474 #endif // DOCTEST_PLATFORM_WINDOWS
06475
06476 // the implementation of parseOption()
06477 bool parseOptionImpl(int argc, const char* const* argv, const char* pattern, String* value) {
06478     // going from the end to the beginning and stopping on the first occurrence from the end
06479     for(int i = argc; i > 0; --i) {
06480         auto index = i - 1;
06481         auto temp = std::strstr(argv[index], pattern);
06482         if(temp && (value || strlen(temp) == strlen(pattern))) {
06483             // eliminate matches in which the chars before the option are not '-'
06484             bool noBadCharsFound = true;
06485             auto curr = argv[index];
06486             while(curr != temp) {
06487                 if(*curr++ != '-') {
06488                     noBadCharsFound = false;
06489                     break;
06490                 }
06491             }
06492             if(noBadCharsFound && argv[index][0] == '-') {
06493                 if(value) {
06494                     // parsing the value of an option
06495                     temp += strlen(pattern);
06496                     const unsigned len = strlen(temp);
06497                     if(len) {
06498                         *value = temp;
06499                         return true;
06500                     }
06501                 } else {
06502                     // just a flag - no value
06503                     return true;
06504                 }
06505             }
06506         }
06507     }
06508     return false;

```

```

06509     }
06510
06511     // parses an option and returns the string after the '=' character
06512     bool parseOption(int argc, const char* const* argv, const char* pattern, String* value = nullptr,
06513                     const String& defaultVal = String()) {
06514         if(value)
06515             *value = defaultVal;
06516 #ifndef DOCTEST_CONFIG_NO_UNPREFIXED_OPTIONS
06517         // offset (normally 3 for "dt-") to skip prefix
06518         if(parseOptionImpl(argc, argv, pattern + strlen(DOCTEST_CONFIG_OPTIONS_PREFIX), value))
06519             return true;
06520 #endif // DOCTEST_CONFIG_NO_UNPREFIXED_OPTIONS
06521         return parseOptionImpl(argc, argv, pattern, value);
06522     }
06523
06524     // locates a flag on the command line
06525     bool parseFlag(int argc, const char* const* argv, const char* pattern) {
06526         return parseOption(argc, argv, pattern);
06527     }
06528
06529     // parses a comma separated list of words after a pattern in one of the arguments in argv
06530     bool parseCommaSepArgs(int argc, const char* const* argv, const char* pattern,
06531                            std::vector<String>& res) {
06532         String filtersString;
06533         if(parseOption(argc, argv, pattern, &filtersString)) {
06534             // tokenize with "," as a separator, unless escaped with backslash
06535             std::ostream s;
06536             auto flush = [&s, &res]() {
06537                 auto string = s.str();
06538                 if(string.size() > 0) {
06539                     res.push_back(string.c_str());
06540                 }
06541                 s.str("");
06542             };
06543
06544             bool seenBackslash = false;
06545             const char* current = filtersString.c_str();
06546             const char* end = current + strlen(current);
06547             while(current != end) {
06548                 char character = *current++;
06549                 if(seenBackslash) {
06550                     seenBackslash = false;
06551                     if(character == ',' || character == '\\') {
06552                         s.put(character);
06553                         continue;
06554                     }
06555                     s.put('\\');
06556                 }
06557                 if(character == '\\') {
06558                     seenBackslash = true;
06559                 } else if(character == ',') {
06560                     flush();
06561                 } else {
06562                     s.put(character);
06563                 }
06564             }
06565
06566             if(seenBackslash) {
06567                 s.put('\\');
06568             }
06569             flush();
06570             return true;
06571         }
06572         return false;
06573     }
06574
06575     enum optionType
06576     {
06577         option_bool,
06578         option_int
06579     };
06580
06581     // parses an int/bool option from the command line
06582     bool parseIntOption(int argc, const char* const* argv, const char* pattern, optionType type,
06583                        int& res) {
06584         String parsedValue;
06585         if(!parseOption(argc, argv, pattern, &parsedValue))
06586             return false;
06587
06588         if(type) {
06589             // integer
06590             // TODO: change this to use std::stoi or something else! currently it uses undefined
06591             behavior - assumes '0' on failed parse...
06592             int theInt = std::atoi(parsedValue.c_str());
06593             if (theInt != 0) {
06594                 res = theInt;
06595                 return true;
06596             }
06597         }
06598     }

```



```

06595     }
06596   } else {
06597       // boolean
06598       const char positive[][5] = { "1", "true", "on", "yes" }; // 5 - strlen("true") + 1
06599       const char negative[][6] = { "0", "false", "off", "no" }; // 6 - strlen("false") + 1
06600
06601       // if the value matches any of the positive/negative possibilities
06602       for (unsigned i = 0; i < 4; i++) {
06603           if (parsedValue.compare(positive[i], true) == 0) {
06604               res = 1;
06605               return true;
06606           }
06607           if (parsedValue.compare(negative[i], true) == 0) {
06608               res = 0;
06609               return true;
06610           }
06611       }
06612   }
06613   return false;
06614 }
06615 } // namespace
06616
06617 Context::Context(int argc, const char* const* argv)
06618     : p(new detail::ContextState) {
06619     parseArgs(argc, argv, true);
06620     if(argc)
06621         p->binary_name = argv[0];
06622 }
06623
06624 Context::~Context() {
06625     if(g_cs == p)
06626         g_cs = nullptr;
06627     delete p;
06628 }
06629
06630 void Context::applyCommandLine(int argc, const char* const* argv) {
06631     parseArgs(argc, argv);
06632     if(argc)
06633         p->binary_name = argv[0];
06634 }
06635
06636 // parses args
06637 void Context::parseArgs(int argc, const char* const* argv, bool withDefaults) {
06638     using namespace detail;
06639
06640     // clang-format off
06641     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "source-file=", p->filters[0]);
06642     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "sf=", p->filters[0]);
06643     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "source-file-exclude=", p->filters[1]);
06644     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "sfe=", p->filters[1]);
06645     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "test-suite=", p->filters[2]);
06646     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "ts=", p->filters[2]);
06647     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "test-suite-exclude=", p->filters[3]);
06648     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "tse=", p->filters[3]);
06649     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "test-case=", p->filters[4]);
06650     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "tc=", p->filters[4]);
06651     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "test-case-exclude=", p->filters[5]);
06652     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "tce=", p->filters[5]);
06653     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "subcase=", p->filters[6]);
06654     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "sc=", p->filters[6]);
06655     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "subcase-exclude=", p->filters[7]);
06656     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "sce=", p->filters[7]);
06657     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "reporters=", p->filters[8]);
06658     parseCommaSepArgs(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "r=", p->filters[8]);
06659     // clang-format on
06660
06661     int intRes = 0;
06662     String strRes;
06663
06664 #define DOCTEST_PARSE_AS_BOOL_OR_FLAG(name, sname, var, default) \
06665     if(parseIntOption(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX name "=", option_bool, intRes) || \
06666         parseIntOption(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX sname "=", option_bool, intRes)) \
06667         p->var = static_cast<bool>(intRes); \
06668     else if(parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX name) || \
06669             parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX sname)) \
06670         p->var = true; \
06671     else if(withDefaults) \
06672         p->var = default;
06673
06674 #define DOCTEST_PARSE_INT_OPTION(name, sname, var, default) \
06675     if(parseIntOption(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX name "=", option_int, intRes) || \
06676         parseIntOption(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX sname "=", option_int, intRes)) \
06677         p->var = intRes; \
06678     else if(withDefaults) \
06679         p->var = default;
06680
06681 #define DOCTEST_PARSE_STR_OPTION(name, sname, var, default) \

```

```

06682     if(parseOption(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX name "=", &strRes, default) || \
06683         parseOption(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX sname "=", &strRes, default) || \
06684         withDefaults)
06685     p->var = strRes
06686
06687     // clang-format off
06688     DOCTEST_PARSE_STR_OPTION("out", "o", out, "");
06689     DOCTEST_PARSE_STR_OPTION("order-by", "ob", order_by, "file");
06690     DOCTEST_PARSE_INT_OPTION("rand-seed", "rs", rand_seed, 0);
06691
06692     DOCTEST_PARSE_INT_OPTION("first", "f", first, 0);
06693     DOCTEST_PARSE_INT_OPTION("last", "l", last, UINT_MAX);
06694
06695     DOCTEST_PARSE_INT_OPTION("abort-after", "aa", abort_after, 0);
06696     DOCTEST_PARSE_INT_OPTION("subcase-filter-levels", "scfl", subcase_filter_levels, INT_MAX);
06697
06698     DOCTEST_PARSE_AS_BOOL_OR_FLAG("success", "s", success, false);
06699     DOCTEST_PARSE_AS_BOOL_OR_FLAG("case-sensitive", "cs", case_sensitive, false);
06700     DOCTEST_PARSE_AS_BOOL_OR_FLAG("exit", "e", exit, false);
06701     DOCTEST_PARSE_AS_BOOL_OR_FLAG("duration", "d", duration, false);
06702     DOCTEST_PARSE_AS_BOOL_OR_FLAG("minimal", "m", minimal, false);
06703     DOCTEST_PARSE_AS_BOOL_OR_FLAG("quiet", "q", quiet, false);
06704     DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-throw", "nt", no_throw, false);
06705     DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-exitcode", "ne", no_exitcode, false);
06706     DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-run", "nr", no_run, false);
06707     DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-intro", "ni", no_intro, false);
06708     DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-version", "nv", no_version, false);
06709     DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-colors", "nc", no_colors, false);
06710     DOCTEST_PARSE_AS_BOOL_OR_FLAG("force-colors", "fc", force_colors, false);
06711     DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-breaks", "nb", no_breaks, false);
06712     DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-skip", "ns", no_skip, false);
06713     DOCTEST_PARSE_AS_BOOL_OR_FLAG("gnu-file-line", "gfl", gnu_file_line, !bool(DOCTEST_MSVC));
06714     DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-path-filenames", "npf", no_path_in_filenames, false);
06715     DOCTEST_PARSE_STR_OPTION("strip-file-prefixes", "sfp", strip_file_prefixes, "");
06716     DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-line-numbers", "nln", no_line_numbers, false);
06717     DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-debug-output", "ndo", no_debug_output, false);
06718     DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-skipped-summary", "nss", no_skipped_summary, false);
06719     DOCTEST_PARSE_AS_BOOL_OR_FLAG("no-time-in-output", "ntio", no_time_in_output, false);
06720     // clang-format on
06721
06722     if(withDefaults) {
06723         p->help = false;
06724         p->version = false;
06725         p->count = false;
06726         p->list_test_cases = false;
06727         p->list_test_suites = false;
06728         p->list_reporters = false;
06729     }
06730     if(parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "help") ||
06731         parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "h") ||
06732         parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "?")) {
06733         p->help = true;
06734         p->exit = true;
06735     }
06736     if(parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "version") ||
06737         parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "v")) {
06738         p->version = true;
06739         p->exit = true;
06740     }
06741     if(parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "count") ||
06742         parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "c")) {
06743         p->count = true;
06744         p->exit = true;
06745     }
06746     if(parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "list-test-cases") ||
06747         parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "ltc")) {
06748         p->list_test_cases = true;
06749         p->exit = true;
06750     }
06751     if(parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "list-test-suites") ||
06752         parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "lts")) {
06753         p->list_test_suites = true;
06754         p->exit = true;
06755     }
06756     if(parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "list-reporters") ||
06757         parseFlag(argc, argv, DOCTEST_CONFIG_OPTIONS_PREFIX "lr")) {
06758         p->list_reporters = true;
06759         p->exit = true;
06760     }
06761 }
06762
06763 // allows the user to add procedurally to the filters from the command line
06764 void Context::addFilter(const char* filter, const char* value) { setOption(filter, value); }
06765
06766 // allows the user to clear all filters from the command line
06767 void Context::clearFilters() {
06768     for(auto& curr : p->filters)

```

```

06769         curr.clear();
06770     }
06771
06772     // allows the user to override procedurally the bool options from the command line
06773     void Context::setOption(const char* option, bool value) {
06774         setOption(option, value ? "true" : "false");
06775     }
06776
06777     // allows the user to override procedurally the int options from the command line
06778     void Context::setOption(const char* option, int value) {
06779         setOption(option, toString(value).c_str());
06780     }
06781
06782     // allows the user to override procedurally the string options from the command line
06783     void Context::setOption(const char* option, const char* value) {
06784         auto argv = String("-") + option + "=" + value;
06785         auto lvalue = argv.c_str();
06786         parseArgs(1, &lvalue);
06787     }
06788
06789     // users should query this in their main() and exit the program if true
06790     bool Context::shouldExit() { return p->exit; }
06791
06792     void Context::setAsDefaultForAssertsOutOfTestCases() { g_cs = p; }
06793
06794     void Context::setAssertHandler(detail::assert_handler ah) { p->ah = ah; }
06795
06796     void Context::setCout(std::ostream* out) { p->cout = out; }
06797
06798     static class DiscardOStream : public std::ostream
06799     {
06800     private:
06801         class : public std::streambuf
06802         {
06803         private:
06804             // allowing some buffering decreases the amount of calls to overflow
06805             char buf[1024];
06806
06807         protected:
06808             std::streamsize xsputn(const char_type*, std::streamsize count) override { return count; }
06809
06810             int_type overflow(int_type ch) override {
06811                 setp(std::begin(buf), std::end(buf));
06812                 return traits_type::not_eof(ch);
06813             }
06814             } discardBuf;
06815
06816     public:
06817         DiscardOStream()
06818             : std::ostream(&discardBuf) {}
06819     } discardOut;
06820
06821     // the main function that does all the filtering and test running
06822     int Context::run() {
06823         using namespace detail;
06824
06825         // save the old context state in case such was setup - for using asserts out of a testing context
06826         auto old_cs = g_cs;
06827         // this is the current contest
06828         g_cs = p;
06829         is_running_in_test = true;
06830
06831         g_no_colors = p->no_colors;
06832         p->resetRunData();
06833
06834         std::fstream fstr;
06835         if(p->cout == nullptr) {
06836             if(p->quiet) {
06837                 p->cout = &discardOut;
06838             } else if(p->out.size()) {
06839                 // to a file if specified
06840                 fstr.open(p->out.c_str(), std::fstream::out);
06841                 p->cout = &fstr;
06842             } else {
06843                 #ifndef DOCTEST_CONFIG_NO_INCLUDE_Iostream
06844                 // stdout by default
06845                 p->cout = &std::cout;
06846                 #else // DOCTEST_CONFIG_NO_INCLUDE_Iostream
06847                 return EXIT_FAILURE;
06848                 #endif // DOCTEST_CONFIG_NO_INCLUDE_Iostream
06849             }
06850         }
06851
06852         FatalConditionHandler::allocateAltStackMem();
06853
06854         auto cleanup_and_return = [&]() {
06855             FatalConditionHandler::freeAltStackMem();

```

```

06856
06857     if(fstr.is_open())
06858         fstr.close();
06859
06860     // restore context
06861     g_cs = old_cs;
06862     is_running_in_test = false;
06863
06864     // we have to free the reporters which were allocated when the run started
06865     for(auto& curr : p->reporters_currently_used)
06866         delete curr;
06867     p->reporters_currently_used.clear();
06868
06869     if(p->numTestCasesFailed && !p->no_exitcode)
06870         return EXIT_FAILURE;
06871     return EXIT_SUCCESS;
06872 };
06873
06874 // setup default reporter if none is given through the command line
06875 if(p->filters[8].empty())
06876     p->filters[8].push_back("console");
06877
06878 // check to see if any of the registered reporters has been selected
06879 for(auto& curr : getReporters()) {
06880     if(matchesAny(curr.first.second.c_str(), p->filters[8], false, p->case_sensitive))
06881         p->reporters_currently_used.push_back(curr.second(*g_cs));
06882 }
06883
06884 // TODO: check if there is nothing in reporters_currently_used
06885
06886 // prepend all listeners
06887 for(auto& curr : getListeners())
06888     p->reporters_currently_used.insert(p->reporters_currently_used.begin(), curr.second(*g_cs));
06889
06890 #ifdef DOCTEST_PLATFORM_WINDOWS
06891     if(isDebuggerActive() && p->no_debug_output == false)
06892         p->reporters_currently_used.push_back(new DebugOutputWindowReporter(*g_cs));
06893 #endif // DOCTEST_PLATFORM_WINDOWS
06894
06895 // handle version, help and no_run
06896 if(p->no_run || p->version || p->help || p->list_reporters) {
06897     DOCTEST_ITERATE_THROUGH_REPORTERS(report_query, QueryData());
06898
06899     return cleanup_and_return();
06900 }
06901
06902 std::vector<const TestCase*> testArray;
06903 for(auto& curr : getRegisteredTests())
06904     testArray.push_back(&curr);
06905 p->numTestCases = testArray.size();
06906
06907 // sort the collected records
06908 if(!testArray.empty()) {
06909     if(p->order_by.compare("file", true) == 0) {
06910         std::sort(testArray.begin(), testArray.end(), fileOrderComparator);
06911     } else if(p->order_by.compare("suite", true) == 0) {
06912         std::sort(testArray.begin(), testArray.end(), suiteOrderComparator);
06913     } else if(p->order_by.compare("name", true) == 0) {
06914         std::sort(testArray.begin(), testArray.end(), nameOrderComparator);
06915     } else if(p->order_by.compare("rand", true) == 0) {
06916         std::srand(p->rand_seed);
06917
06918         // random_shuffle implementation
06919         const auto first = &testArray[0];
06920         for(size_t i = testArray.size() - 1; i > 0; --i) {
06921             int idxToSwap = std::rand() % (i + 1);
06922
06923             const auto temp = first[i];
06924
06925             first[i] = first[idxToSwap];
06926             first[idxToSwap] = temp;
06927         }
06928     } else if(p->order_by.compare("none", true) == 0) {
06929         // means no sorting - beneficial for death tests which call into the executable
06930         // with a specific test case in mind - we don't want to slow down the startup times
06931     }
06932 }
06933
06934 std::set<String> testSuitesPassingFilter;
06935
06936 bool query_mode = p->count || p->list_test_cases ||
p->list_test_suites;
06937 std::vector<const TestCaseData*> queryResults;
06938
06939 if(!query_mode)
06940     DOCTEST_ITERATE_THROUGH_REPORTERS(test_run_start, DOCTEST_EMPTY);
06941

```

```

06942 // invoke the registered functions if they match the filter criteria (or just count them)
06943 for(auto& curr : testArray) {
06944     const auto& tc = *curr;
06945
06946     bool skip_me = false;
06947     if(tc.m_skip && !p->no_skip)
06948         skip_me = true;
06949
06950     if(!matchesAny(tc.m_file.c_str(), p->filters[0], true, p->case_sensitive))
06951         skip_me = true;
06952     if(matchesAny(tc.m_file.c_str(), p->filters[1], false, p->case_sensitive))
06953         skip_me = true;
06954     if(!matchesAny(tc.m_test_suite, p->filters[2], true, p->case_sensitive))
06955         skip_me = true;
06956     if(matchesAny(tc.m_test_suite, p->filters[3], false, p->case_sensitive))
06957         skip_me = true;
06958     if(!matchesAny(tc.m_name, p->filters[4], true, p->case_sensitive))
06959         skip_me = true;
06960     if(matchesAny(tc.m_name, p->filters[5], false, p->case_sensitive))
06961         skip_me = true;
06962
06963     if(!skip_me)
06964         p->numTestCasesPassingFilters++;
06965
06966     // skip the test if it is not in the execution range
06967     if((p->last < p->numTestCasesPassingFilters && p->first <= p->last) ||
06968        (p->first > p->numTestCasesPassingFilters))
06969         skip_me = true;
06970
06971     if(skip_me) {
06972         if(!query_mode)
06973             DOCTEST_ITERATE_THROUGH_REPORTERS(test_case_skipped, tc);
06974         continue;
06975     }
06976
06977     // do not execute the test if we are to only count the number of filter passing tests
06978     if(p->count)
06979         continue;
06980
06981     // print the name of the test and don't execute it
06982     if(p->list_test_cases) {
06983         queryResults.push_back(&tc);
06984         continue;
06985     }
06986
06987     // print the name of the test suite if not done already and don't execute it
06988     if(p->list_test_suites) {
06989         if((testSuitesPassingFilt.count(tc.m_test_suite) == 0) && tc.m_test_suite[0] != '\0') {
06990             queryResults.push_back(&tc);
06991             testSuitesPassingFilt.insert(tc.m_test_suite);
06992             p->numTestSuitesPassingFilters++;
06993         }
06994         continue;
06995     }
06996
06997     // execute the test if it passes all the filtering
06998     {
06999         p->currentTest = &tc;
07000
07001         p->failure_flags = TestCaseFailureReason::None;
07002         p->seconds = 0;
07003
07004         // reset atomic counters
07005         p->numAssertsFailedCurrentTest_atomic = 0;
07006         p->numAssertsCurrentTest_atomic = 0;
07007
07008         p->fullyTraversedSubcases.clear();
07009
07010         DOCTEST_ITERATE_THROUGH_REPORTERS(test_case_start, tc);
07011
07012         p->timer.start();
07013
07014         bool run_test = true;
07015
07016         do {
07017             // reset some of the fields for subcases (except for the set of fully passed ones)
07018             p->reachedLeaf = false;
07019             // May not be empty if previous subcase exited via exception.
07020             p->subcaseStack.clear();
07021             p->currentSubcaseDepth = 0;
07022
07023             p->shouldLogCurrentException = true;
07024
07025             // reset stuff for logging with INFO()
07026             p->stringifiedContexts.clear();
07027
07028 #ifndef DOCTEST_CONFIG_NO_EXCEPTIONS

```

```

07029         try {
07030 #endif // DOCTEST_CONFIG_NO_EXCEPTIONS
07031 // MSVC 2015 diagnoses fatalConditionHandler as unused (because reset() is a static method)
07032 DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4101) // unreferenced local variable
07033         FatalConditionHandler fatalConditionHandler; // Handle signals
07034         // execute the test
07035         tc.m_test();
07036         fatalConditionHandler.reset();
07037 DOCTEST_MSVC_SUPPRESS_WARNING_POP
07038 #ifndef DOCTEST_CONFIG_NO_EXCEPTIONS
07039         } catch(const TestFailureException&) {
07040             p->failure_flags |= TestCaseFailureReason::AssertFailure;
07041         } catch(...) {
07042             DOCTEST_ITERATE_THROUGH_REPORTERS(test_case_exception,
07043                                             {translateActiveException(), false});
07044             p->failure_flags |= TestCaseFailureReason::Exception;
07045         }
07046 #endif // DOCTEST_CONFIG_NO_EXCEPTIONS
07047
07048         // exit this loop if enough assertions have failed - even if there are more subcases
07049         if(p->abort_after > 0 &&
07050            p->numAssertsFailed + p->numAssertsFailedCurrentTest_atomic >= p->abort_after) {
07051             run_test = false;
07052             p->failure_flags |= TestCaseFailureReason::TooManyFailedAsserts;
07053         }
07054
07055         if(!p->nextSubcaseStack.empty() && run_test)
07056             DOCTEST_ITERATE_THROUGH_REPORTERS(test_case_reenter, tc);
07057         if(p->nextSubcaseStack.empty())
07058             run_test = false;
07059     } while(run_test);
07060
07061     p->finalizeTestCaseData();
07062
07063     DOCTEST_ITERATE_THROUGH_REPORTERS(test_case_end, *g_cs);
07064
07065     p->currentTest = nullptr;
07066
07067     // stop executing tests if enough assertions have failed
07068     if(p->abort_after > 0 && p->numAssertsFailed >= p->abort_after)
07069         break;
07070 }
07071 }
07072
07073 if(!query_mode) {
07074     DOCTEST_ITERATE_THROUGH_REPORTERS(test_run_end, *g_cs);
07075 } else {
07076     QueryData qdata;
07077     qdata.run_stats = g_cs;
07078     qdata.data = queryResults.data();
07079     qdata.num_data = unsigned(queryResults.size());
07080     DOCTEST_ITERATE_THROUGH_REPORTERS(report_query, qdata);
07081 }
07082
07083 return cleanup_and_return();
07084 }
07085
07086 DOCTEST_DEFINE_INTERFACE(IReporter)
07087
07088 int IReporter::get_num_active_contexts() { return detail::g_infoContexts.size(); }
07089 const IContextScope* IReporter::get_active_contexts() {
07090     return get_num_active_contexts() ? &detail::g_infoContexts[0] : nullptr;
07091 }
07092
07093 int IReporter::get_num_stringified_contexts() { return detail::g_cs->stringifiedContexts.size(); }
07094 const String* IReporter::get_stringified_contexts() {
07095     return get_num_stringified_contexts() ? &detail::g_cs->stringifiedContexts[0] : nullptr;
07096 }
07097
07098 namespace detail {
07099     void registerReporterImpl(const char* name, int priority, reporterCreatorFunc c, bool isReporter)
07100     {
07101         if(isReporter)
07102             getReporters().insert(reporterMap::value_type(reporterMap::key_type(priority, name), c));
07103         else
07104             getListeners().insert(reporterMap::value_type(reporterMap::key_type(priority, name), c));
07105     }
07106 } // namespace detail
07107 } // namespace doctest
07108
07109 #endif // DOCTEST_CONFIG_DISABLE
07110
07111 #ifdef DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
07112 DOCTEST_MSVC_SUPPRESS_WARNING_WITH_PUSH(4007) // 'function' : must be 'attribute' - see issue #182
07113 int main(int argc, char** argv) { return doctest::Context(argc, argv).run(); }
07114 DOCTEST_MSVC_SUPPRESS_WARNING_POP

```

```

07115 #endif // DOCTEST_CONFIG_IMPLEMENT_WITH_MAIN
07116
07117 DOCTEST_CLANG_SUPPRESS_WARNING_POP
07118 DOCTEST_MSVC_SUPPRESS_WARNING_POP
07119 DOCTEST_GCC_SUPPRESS_WARNING_POP
07120
07121 DOCTEST_SUPPRESS_COMMON_WARNINGS_POP
07122
07123 #endif // DOCTEST_LIBRARY_IMPLEMENTATION
07124 #endif // DOCTEST_CONFIG_IMPLEMENT
07125
07126 #ifdef DOCTEST_UNDEF_WIN32_LEAN_AND_MEAN
07127 #undef WIN32_LEAN_AND_MEAN
07128 #undef DOCTEST_UNDEF_WIN32_LEAN_AND_MEAN
07129 #endif // DOCTEST_UNDEF_WIN32_LEAN_AND_MEAN
07130
07131 #ifdef DOCTEST_UNDEF_NOMINMAX
07132 #undef NOMINMAX
07133 #undef DOCTEST_UNDEF_NOMINMAX
07134 #endif // DOCTEST_UNDEF_NOMINMAX

```

## 6.4 vector.h

```

00001 #ifndef VECTOR_H
00002 #define VECTOR_H
00003
00004 #include <cstdint>
00005 #include <limits>
00006 #include <utility>
00007 #include <algorithm>
00008 #include <stdexcept>
00009 #include <initializer_list>
00010 #include <iostream>
00011
00012 template <typename T>
00013 class Vector {
00014 private:
00015     T* vec_;
00016     size_t capacity_;
00017     size_t size_;
00018     size_t reallocations; // Skaitiklis perskirstymams
00019
00020     void reallocate(size_t new_capacity) {
00021         T* new_vec = new T[new_capacity];
00022         try {
00023             std::move(vec_, vec_ + size_, new_vec);
00024         } catch (...) {
00025             delete[] new_vec;
00026             throw;
00027         }
00028         delete[] vec_;
00029         vec_ = new_vec;
00030         capacity_ = new_capacity;
00031         reallocations++;
00032     }
00033
00034 public:
00035     // Member types
00036     using value_type = T;
00037     using size_type = size_t;
00038     using reference = T&;
00039     using const_reference = const T&;
00040     using iterator = T*;
00041     using const_iterator = const T*;
00042     using reverse_iterator = std::reverse_iterator<iterator>;
00043     using const_reverse_iterator = std::reverse_iterator<const_iterator>;
00044
00045     // Constructors
00046     Vector() : vec_(nullptr), capacity_(0), size_(0), reallocations(0) {}
00047
00048     explicit Vector(size_type count)
00049         : vec_(new T[count]), capacity_(count), size_(count), reallocations(0) {
00050         std::fill_n(vec_, size_, T());
00051     }
00052
00053     Vector(size_type count, const T& value)
00054         : vec_(new T[count]), capacity_(count), size_(count), reallocations(0) {
00055         std::fill_n(vec_, size_, value);
00056     }
00057
00058     Vector(std::initializer_list<T> list)
00059         : vec_(new T[list.size()]), capacity_(list.size()), size_(list.size()), reallocations(0) {
00060         std::copy(list.begin(), list.end(), vec_);

```

```

00061     }
00062
00063     // Copy constructor
00064     Vector(const Vector& other)
00065         : vec_(new T[other.capacity_]), capacity_(other.capacity_), size_(other.size_),
00066         reallocations(0) {
00067         std::copy(other.vec_, other.vec_ + size_, vec_);
00068     }
00069
00070     // Move constructor
00071     Vector(Vector&& other) noexcept
00072         : vec_(other.vec_), capacity_(other.capacity_), size_(other.size_),
00073         reallocations(other.reallocations) {
00074         other.vec_ = nullptr;
00075         other.capacity_ = 0;
00076         other.size_ = 0;
00077         other.reallocations = 0;
00078     }
00079
00080     // Destructor
00081     ~Vector() {
00082         delete[] vec_;
00083     }
00084
00085     // Assignment operators
00086     Vector& operator=(const Vector& other) {
00087         if (this != &other) {
00088             Vector temp(other);
00089             swap(temp);
00090         }
00091         return *this;
00092     }
00093
00094     Vector& operator=(Vector&& other) noexcept {
00095         if (this != &other) {
00096             delete[] vec_;
00097             vec_ = other.vec_;
00098             size_ = other.size_;
00099             capacity_ = other.capacity_;
00100             reallocations = other.reallocations;
00101             other.vec_ = nullptr;
00102             other.size_ = 0;
00103             other.capacity_ = 0;
00104             other.reallocations = 0;
00105         }
00106         return *this;
00107     }
00108
00109     // Element access (NEKEISTA)
00110     reference operator[](size_type pos) { return vec_[pos]; }
00111     const_reference operator[](size_type pos) const { return vec_[pos]; }
00112     reference at(size_type pos) {
00113         if (pos >= size_) throw std::out_of_range("out of range");
00114         return vec_[pos];
00115     }
00116     const_reference at(size_type pos) const {
00117         if (pos >= size_) throw std::out_of_range("out of range");
00118         return vec_[pos];
00119     }
00120     reference front() { return vec_[0]; }
00121     const_reference front() const { return vec_[0]; }
00122     reference back() { return vec_[size_ - 1]; }
00123     const_reference back() const { return vec_[size_ - 1]; }
00124     T* data() noexcept { return vec_; }
00125     const T* data() const noexcept { return vec_; }
00126
00127     // Iterators (NEKEISTA)
00128     iterator begin() noexcept { return vec_; }
00129     const_iterator begin() const noexcept { return vec_; }
00130     const_iterator cbegin() const noexcept { return vec_; }
00131     iterator end() noexcept { return vec_ + size_; }
00132     const_iterator end() const noexcept { return vec_ + size_; }
00133     const_iterator cend() const noexcept { return vec_ + size_; }
00134     reverse_iterator rbegin() noexcept { return reverse_iterator(end()); }
00135     const_reverse_iterator rbegin() const noexcept { return const_reverse_iterator(end()); }
00136     const_reverse_iterator crbegin() const noexcept { return const_reverse_iterator(end()); }
00137     reverse_iterator rend() noexcept { return reverse_iterator(begin()); }
00138     const_reverse_iterator rend() const noexcept { return const_reverse_iterator(begin()); }
00139     const_reverse_iterator crend() const noexcept { return const_reverse_iterator(begin()); }
00140
00141     // Capacity
00142     bool empty() const noexcept { return size_ == 0; }
00143     size_type size() const noexcept { return size_; }
00144     size_type capacity() const noexcept { return capacity_; }
00145     size_type getReallocations() const { return reallocations; }
00146     size_type max_size() const noexcept { return std::numeric_limits<size_type>::max() / sizeof(T); }

```



```

00146     void reserve(size_type new_cap) {
00147         if (new_cap > capacity_) {
00148             reallocate(new_cap);
00149         }
00150     }
00151
00152     void shrink_to_fit() {
00153         if (size_ < capacity_) {
00154             reallocate(size_);
00155         }
00156     }
00157
00158     // Modifiers
00159     void clear() noexcept { size_ = 0; }
00160
00161     iterator insert(const_iterator pos, const T& value) {
00162         return emplace(pos, value);
00163     }
00164
00165     iterator insert(const_iterator pos, T&& value) {
00166         return emplace(pos, std::move(value));
00167     }
00168
00169     iterator erase(const_iterator pos) {
00170         if (pos < cbegin() || pos >= cend()) throw std::out_of_range("out of range");
00171         iterator non_const_pos = begin() + (pos - cbegin());
00172         std::move(non_const_pos + 1, end(), non_const_pos);
00173         --size_;
00174         return non_const_pos;
00175     }
00176
00177     iterator erase(const_iterator first, const_iterator last) {
00178         if (first < cbegin() || last > cend() || first > last) throw std::out_of_range("out of
range");
00179         iterator non_const_first = begin() + (first - cbegin());
00180         iterator non_const_last = begin() + (last - cbegin());
00181         std::move(non_const_last, end(), non_const_first);
00182         size_ -= (last - first);
00183         return non_const_first;
00184     }
00185
00186     void push_back(const T& value) {
00187         if (size_ == capacity_) reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00188         vec_[size_++] = value;
00189     }
00190
00191     void push_back(T&& value) {
00192         if (size_ == capacity_) reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00193         vec_[size_++] = std::move(value);
00194     }
00195
00196     template <typename... Args>
00197     reference emplace_back(Args&&... args) {
00198         if (size_ >= capacity_) reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00199         new (&vec_[size_]) T(std::forward<Args>(args)...);
00200         return vec_[size_++];
00201     }
00202
00203     template <typename... Args>
00204     iterator emplace(const_iterator pos, Args&&... args) {
00205         if (pos < cbegin() || pos > cend()) throw std::out_of_range("Vector::emplace - iterator out of
range");
00206         size_type offset = pos - cbegin();
00207         if (size_ >= capacity_) reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00208         iterator insert_pos = begin() + offset;
00209         if (insert_pos != end()) {
00210             std::move_backward(insert_pos, end(), end() + 1);
00211         }
00212         new (&(*insert_pos)) T(std::forward<Args>(args)...);
00213         ++size_;
00214         return insert_pos;
00215     }
00216
00217     void pop_back() {
00218         if (size_ > 0) --size_;
00219     }
00220
00221     void resize(size_type count) {
00222         if (count > capacity_) reserve(count);
00223         if (count > size_) std::fill(vec_ + size_, vec_ + count, T());
00224         size_ = count;
00225     }
00226
00227     void resize(size_type count, const value_type& value) {
00228         if (count > capacity_) reserve(count);
00229         if (count > size_) std::fill(vec_ + size_, vec_ + count, value);
00230         size_ = count;

```

```

00231     }
00232
00233     void swap(Vector& other) noexcept {
00234         std::swap(vec_, other.vec_);
00235         std::swap(size_, other.size_);
00236         std::swap(capacity_, other.capacity_);
00237         std::swap(reallocations, other.reallocations);
00238     }
00239
00240     // Comparison operators (NEKEISTA)
00241     bool operator==(const Vector& other) const {
00242         if (size_ != other.size_) return false;
00243         return std::equal(begin(), end(), other.begin());
00244     }
00245
00246     bool operator!=(const Vector& other) const { return !(*this == other); }
00247     bool operator<(const Vector& other) const {
00248         return std::lexicographical_compare(begin(), end(), other.begin(), other.end());
00249     }
00250     bool operator<=(const Vector& other) const { return !(*this > other); }
00251     bool operator>(const Vector& other) const { return other < *this; }
00252     bool operator>=(const Vector& other) const { return !(*this < other); }
00253 };
00254
00255 // Non-member swap function
00256 template <typename T>
00257 void swap(Vector<T>& lhs, Vector<T>& rhs) noexcept {
00258     lhs.swap(rhs);
00259 }
00260
00261 #endif // VECTOR_H

```

## 6.5 vector.h

```

00001 #ifndef VECTOR_H
00002 #define VECTOR_H
00003
00004 #include <cstdint>
00005 #include <limits>
00006 #include <utility>
00007 #include <algorithm>
00008 #include <stdexcept>
00009 #include <initializer_list>
00010 #include <iostream>
00011
00012 template <typename T>
00013 class Vector {
00014 private:
00015     T* vec_;
00016     size_t capacity_;
00017     size_t size_;
00018     size_t reallocations; // Skaitiklis perskirstymams
00019
00020     void reallocate(size_t new_capacity) {
00021         T* new_vec = new T[new_capacity];
00022         try {
00023             std::move(vec_, vec_ + size_, new_vec);
00024         } catch (...) {
00025             delete[] new_vec;
00026             throw;
00027         }
00028         delete[] vec_;
00029         vec_ = new_vec;
00030         capacity_ = new_capacity;
00031         reallocations++;
00032     }
00033
00034 public:
00035     // Member types
00036     using value_type = T;
00037     using size_type = size_t;
00038     using reference = T&
00039     using const_reference = const T&
00040     using iterator = T*;
00041     using const_iterator = const T*;
00042     using reverse_iterator = std::reverse_iterator<iterator>;
00043     using const_reverse_iterator = std::reverse_iterator<const_iterator>;
00044
00045     // Constructors
00046     Vector() : vec_(nullptr), capacity_(0), size_(0), reallocations(0) {}
00047
00048     explicit Vector(size_type count)
00049         : vec_(new T[count]), capacity_(count), size_(count), reallocations(0) {

```

```

00050         std::fill_n(vec_, size_, T());
00051     }
00052
00053     Vector(size_type count, const T& value)
00054         : vec_(new T[count]), capacity_(count), size_(count), reallocations(0) {
00055         std::fill_n(vec_, size_, value);
00056     }
00057
00058     Vector(std::initializer_list<T> list)
00059         : vec_(new T[list.size()]), capacity_(list.size()), size_(list.size()), reallocations(0) {
00060         std::copy(list.begin(), list.end(), vec_);
00061     }
00062
00063     // Copy constructor
00064     Vector(const Vector& other)
00065         : vec_(new T[other.capacity_]), capacity_(other.capacity_), size_(other.size_),
00066         reallocations(0) {
00067         std::copy(other.vec_, other.vec_ + size_, vec_);
00068     }
00069
00070     // Move constructor
00071     Vector(Vector&& other) noexcept
00072         : vec_(other.vec_), capacity_(other.capacity_), size_(other.size_),
00073         reallocations(other.reallocations) {
00074         other.vec_ = nullptr;
00075         other.capacity_ = 0;
00076         other.size_ = 0;
00077         other.reallocations = 0;
00078     }
00079
00080     // Destructor
00081     ~Vector() {
00082         delete[] vec_;
00083     }
00084
00085     // Assignment operators
00086     Vector& operator=(const Vector& other) {
00087         if (this != &other) {
00088             Vector temp(other);
00089             swap(temp);
00090         }
00091         return *this;
00092     }
00093
00094     Vector& operator=(Vector&& other) noexcept {
00095         if (this != &other) {
00096             delete[] vec_;
00097             vec_ = other.vec_;
00098             size_ = other.size_;
00099             capacity_ = other.capacity_;
00100             reallocations = other.reallocations;
00101             other.vec_ = nullptr;
00102             other.size_ = 0;
00103             other.capacity_ = 0;
00104             other.reallocations = 0;
00105         }
00106         return *this;
00107     }
00108
00109     // Element access (NEKEISTA)
00110     reference operator[](size_type pos) { return vec_[pos]; }
00111     const_reference operator[](size_type pos) const { return vec_[pos]; }
00112     reference at(size_type pos) {
00113         if (pos >= size_) throw std::out_of_range("out of range");
00114         return vec_[pos];
00115     }
00116     const_reference at(size_type pos) const {
00117         if (pos >= size_) throw std::out_of_range("out of range");
00118         return vec_[pos];
00119     }
00120     reference front() { return vec_[0]; }
00121     const_reference front() const { return vec_[0]; }
00122     reference back() { return vec_[size_ - 1]; }
00123     const_reference back() const { return vec_[size_ - 1]; }
00124     T* data() noexcept { return vec_; }
00125     const T* data() const noexcept { return vec_; }
00126
00127     // Iterators (NEKEISTA)
00128     iterator begin() noexcept { return vec_; }
00129     const_iterator begin() const noexcept { return vec_; }
00130     const_iterator cbegin() const noexcept { return vec_; }
00131     iterator end() noexcept { return vec_ + size_; }
00132     const_iterator end() const noexcept { return vec_ + size_; }
00133     reverse_iterator rbegin() noexcept { return reverse_iterator(end()); }
00134     const_reverse_iterator rbegin() const noexcept { return const_reverse_iterator(end()); }
00135     const_reverse_iterator crbegin() const noexcept { return const_reverse_iterator(end()); }

```

```

00135     reverse_iterator rend() noexcept { return reverse_iterator(begin()); }
00136     const_reverse_iterator rend() const noexcept { return const_reverse_iterator(begin()); }
00137     const_reverse_iterator crend() const noexcept { return const_reverse_iterator(begin()); }
00138
00139     // Capacity
00140     bool empty() const noexcept { return size_ == 0; }
00141     size_type size() const noexcept { return size_; }
00142     size_type capacity() const noexcept { return capacity_; }
00143     size_type getReallocations() const { return reallocations; }
00144     size_type max_size() const noexcept { return std::numeric_limits<size_type>::max() / sizeof(T); }
00145
00146     void reserve(size_type new_cap) {
00147         if (new_cap > capacity_) {
00148             reallocate(new_cap);
00149         }
00150     }
00151
00152     void shrink_to_fit() {
00153         if (size_ < capacity_) {
00154             reallocate(size_);
00155         }
00156     }
00157
00158     // Modifiers
00159     void clear() noexcept { size_ = 0; }
00160
00161     iterator insert(const_iterator pos, const T& value) {
00162         return emplace(pos, value);
00163     }
00164
00165     iterator insert(const_iterator pos, T&& value) {
00166         return emplace(pos, std::move(value));
00167     }
00168
00169     iterator erase(const_iterator pos) {
00170         if (pos < cbegin() || pos >= cend()) throw std::out_of_range("out of range");
00171         iterator non_const_pos = begin() + (pos - cbegin());
00172         std::move(non_const_pos + 1, end(), non_const_pos);
00173         --size_;
00174         return non_const_pos;
00175     }
00176
00177     iterator erase(const_iterator first, const_iterator last) {
00178         if (first < cbegin() || last > cend() || first > last) throw std::out_of_range("out of
00179 range");
00180         iterator non_const_first = begin() + (first - cbegin());
00181         iterator non_const_last = begin() + (last - cbegin());
00182         std::move(non_const_last, end(), non_const_first);
00183         size_ -= (last - first);
00184         return non_const_first;
00185     }
00186
00187     void push_back(const T& value) {
00188         if (size_ == capacity_) reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00189         vec_[size_++] = value;
00190     }
00191
00192     void push_back(T&& value) {
00193         if (size_ == capacity_) reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00194         vec_[size_++] = std::move(value);
00195     }
00196
00197     template <typename... Args>
00198     reference emplace_back(Args&&... args) {
00199         if (size_ >= capacity_) reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00200         new (&vec_[size_]) T(std::forward<Args>(args)...);
00201         return vec_[size_++];
00202     }
00203
00204     template <typename... Args>
00205     iterator emplace(const_iterator pos, Args&&... args) {
00206         if (pos < cbegin() || pos > cend()) throw std::out_of_range("Vector::emplace - iterator out of
00207 range");
00208         size_type offset = pos - cbegin();
00209         if (size_ >= capacity_) reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00210         iterator insert_pos = begin() + offset;
00211         if (insert_pos != end()) {
00212             std::move_backward(insert_pos, end(), end() + 1);
00213         }
00214         new (&(*insert_pos)) T(std::forward<Args>(args)...);
00215         ++size_;
00216         return insert_pos;
00217     }
00218
00219     void pop_back() {
00220         if (size_ > 0) --size_;
00221     }

```

```

00220
00221 void resize(size_type count) {
00222     if (count > capacity_) reserve(count);
00223     if (count > size_) std::fill(vec_ + size_, vec_ + count, T());
00224     size_ = count;
00225 }
00226
00227 void resize(size_type count, const value_type& value) {
00228     if (count > capacity_) reserve(count);
00229     if (count > size_) std::fill(vec_ + size_, vec_ + count, value);
00230     size_ = count;
00231 }
00232
00233 void swap(Vector& other) noexcept {
00234     std::swap(vec_, other.vec_);
00235     std::swap(size_, other.size_);
00236     std::swap(capacity_, other.capacity_);
00237     std::swap(reallocations, other.reallocations);
00238 }
00239
00240 // Comparison operators (NEKEISTA)
00241 bool operator==(const Vector& other) const {
00242     if (size_ != other.size_) return false;
00243     return std::equal(begin(), end(), other.begin());
00244 }
00245
00246 bool operator!=(const Vector& other) const { return !(*this == other); }
00247 bool operator<(const Vector& other) const {
00248     return std::lexicographical_compare(begin(), end(), other.begin(), other.end());
00249 }
00250 bool operator<=(const Vector& other) const { return !(*this > other); }
00251 bool operator>(const Vector& other) const { return other < *this; }
00252 bool operator>=(const Vector& other) const { return !(*this < other); }
00253 };
00254
00255 // Non-member swap function
00256 template <typename T>
00257 void swap(Vector<T>& lhs, Vector<T>& rhs) noexcept {
00258     lhs.swap(rhs);
00259 }
00260
00261 #endif // VECTOR_H

```

## 6.6 zmogus.h

```

00001 #ifndef ZMOGUS_H
00002 #define ZMOGUS_H
00003
00004 #include <string>
00005
00006 class Zmogus {
00007 protected:
00008     std::string vardas_;
00009     std::string pavarde_;
00010
00011 public:
00012     Zmogus() = default;
00013     Zmogus(const std::string& vardas, const std::string& pavarde)
00014         : vardas_(vardas), pavarde_(pavarde) {}
00015
00016     virtual ~Zmogus() = default;
00017
00018     std::string vardas() const { return vardas_; }
00019     std::string pavarde() const { return pavarde_; }
00020
00021     void setVardas(const std::string& vardas) { vardas_ = vardas; }
00022     void setPavarde(const std::string& pavarde) { pavarde_ = pavarde; }
00023
00024     virtual std::ostream& spausdinti(std::ostream& os) const = 0;
00025 };
00026
00027 #endif

```



# Index

3.0 nuosavos Vector klasės testavimas, [1](#)

`doctest::Approx`, [9](#)

`doctest::AssertData`, [10](#)

`doctest::AssertData::StringContains`, [31](#)

`doctest::Contains`, [11](#)

`doctest::Context`, [11](#)

`doctest::ContextOptions`, [12](#)

`doctest::CurrentTestCaseStats`, [14](#)

`doctest::detail::ContextScope` `< L >`, [13](#)

[stringify](#), [13](#)

`doctest::detail::ContextScopeBase`, [14](#)

`doctest::detail::deferred_false` `< T >`, [15](#)

`doctest::detail::ExceptionTranslator` `< T >`, [16](#)

[translate](#), [16](#)

`doctest::detail::Expression_lhs` `< L >`, [16](#)

[operator Result](#), [17](#)

`doctest::detail::ExpressionDecomposer`, [17](#)

`doctest::detail::filldata` `< const char[N]>`, [18](#)

`doctest::detail::filldata` `< const void * >`, [18](#)

`doctest::detail::filldata` `< T >`, [18](#)

`doctest::detail::filldata` `< T * >`, [19](#)

`doctest::detail::filldata` `< T[N]>`, [19](#)

`doctest::detail::has_insertion_operator` `< T, decltype(operator<<(declval<`

[std::ostream & >\(\)\)](#), `declval< const T & >())`,

[void\(\)>](#), [20](#)

`doctest::detail::has_insertion_operator` `< T, typename`  
`>`, [19](#)

`doctest::detail::IExceptionTranslator`, [21](#)

`doctest::detail::MessageBuilder`, [25](#)

`doctest::detail::RelationalComparator` `< int, L, R >`, [27](#)

`doctest::detail::Result`, [28](#)

`doctest::detail::ResultBuilder`, [29](#)

[unary\\_assert](#), [30](#)

`doctest::detail::should_stringify_as_underlying_type` `< T`  
`>`, [30](#)

`doctest::detail::StringMakerBase` `< C >`, [32](#)

`doctest::detail::StringMakerBase` `< true >`, [32](#)

`doctest::detail::Subcase`, [33](#)

`doctest::detail::TestCase`, [34](#)

`doctest::detail::TestFailureException`, [36](#)

`doctest::detail::TestSuite`, [36](#)

`doctest::detail::types::enable_if` `< COND, T >`, [15](#)

`doctest::detail::types::enable_if` `< true, T >`, [15](#)

`doctest::detail::types::false_type`, [17](#)

`doctest::detail::types::is_array` `< T >`, [22](#)

`doctest::detail::types::is_array` `< T[SIZE]>`, [22](#)

`doctest::detail::types::is_enum` `< T >`, [23](#)

`doctest::detail::types::is_pointer` `< T >`, [23](#)

`doctest::detail::types::is_pointer` `< T * >`, [23](#)

`doctest::detail::types::is_rvalue_reference` `< T >`, [24](#)

`doctest::detail::types::is_rvalue_reference` `< T && >`, [24](#)

`doctest::detail::types::remove_const` `< const T >`, [27](#)

`doctest::detail::types::remove_const` `< T >`, [27](#)

`doctest::detail::types::remove_reference` `< T >`, [27](#)

`doctest::detail::types::remove_reference` `< T & >`, [28](#)

`doctest::detail::types::remove_reference` `< T && >`, [28](#)

`doctest::detail::types::true_type`, [37](#)

`doctest::detail::types::underlying_type` `< T >`, [37](#)

`doctest::IContextScope`, [20](#)

`doctest::IReporter`, [21](#)

`doctest::IsNaN` `< F >`, [25](#)

`doctest::MessageData`, [26](#)

`doctest::QueryData`, [26](#)

`doctest::String`, [30](#)

`doctest::StringMaker` `< T >`, [31](#)

`doctest::SubcaseSignature`, [34](#)

`doctest::TestCaseData`, [35](#)

`doctest::TestCaseException`, [36](#)

`doctest::TestRunStats`, [36](#)

`operator Result`

[doctest::detail::Expression\\_lhs `< L >`, \[17\]\(#\)](#)

[spausdinti](#)

[Studentas](#), [33](#)

`std::basic_istream` `< charT, traits >`, [10](#)

`std::basic_ostream` `< charT, traits >`, [11](#)

`std::char_traits` `< charT >`, [11](#)

`std::tuple` `< Types >`, [37](#)

`stringify`

[doctest::detail::ContextScope `< L >`, \[13\]\(#\)](#)

`Studentas`, [32](#)

[spausdinti](#), [33](#)

`Testavimas/doctest.h`, [44](#)

`Testavimas/vector.h`, [129](#)

`translate`

[doctest::detail::ExceptionTranslator `< T >`, \[16\]\(#\)](#)

`unary_assert`

[doctest::detail::ResultBuilder](#), [30](#)

`Vector` `< T >`, [38](#)

`Zmogus`, [40](#)