Introduction to Parallel Computation - Project 2

Generated by Doxygen 1.8.15

Chapter 1

Introduction to Paraller Programming - Project #2

This is the second project for the Introduction to Paraller Programming class of the Information and Computers Engineering curriculum at the University of West Attica.

The class is about, well, parallel programming using MPI.

The project's tasks are as follows:

- [x] Read a square matrix from the user.
- [x] Check if it is `strictly diagonally dominant` and print a message to the screen.

If it is:

- [x] Find the maximum element of the diagonal and print it to the screen.
- [x] Calculate and print a matrix based on the following formulas:

where m is the previously calculated maximum element of the diagonal and A the original matrix.

- [x] Find the minimum element of the new matrix and print it to the screen with it's position in the matrix.

The project has to be implemented using MPI's Collective Communication methods.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

Chapter 5

Class Documentation

5.1 Catch::Detail::Approx Class Reference

Public Member Functions

- **Approx** (double value)
- Approx **operator-** () const
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> Approx **operator()** (T const &value)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> **Approx** (T const &value)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> Approx & **epsilon** (T const &newEpsilon)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> Approx & **margin** (T const &newMargin)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> Approx & **scale** (T const &newScale)
- std::string **toString** () const

Static Public Member Functions

- static Approx **custom** ()

Friends

- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> bool **operator==** (const T &lhs, Approx const &rhs)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> bool **operator==** (Approx const &lhs, const T &rhs)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> bool **operator!=** (T const &lhs, Approx const &rhs)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> bool **operator!=** (Approx const &lhs, T const &rhs)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> bool **operator<=** (T const &lhs, Approx const &rhs)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> bool **operator<=** (Approx const &lhs, T const &rhs)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> bool **operator>=** (T const &lhs, Approx const &rhs)
- template<typename T , typename = typename std::enable_if<std::is_constructible<double, T>::value>::type> bool **operator>=** (Approx const &lhs, T const &rhs)

5.1.1 Detailed Description

Definition at line 2470 of file catch.hpp.

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

- lib/catch.hpp

5.2 Catch::AssertionHandler Class Reference

Public Member Functions

- **AssertionHandler** (StringRef const &macroName, SourceLineInfo const &lineInfo, StringRef captured↩ Expression, ResultDisposition::Flags resultDisposition)
- template<typename T >
  void **handleExpr** (ExprLhs< T > const &expr)
- void **handleExpr** (ITransientExpression const &expr)
- void **handleMessage** (ResultWas::OfType resultType, StringRef const &message)
- void **handleExceptionThrownAsExpected** ()
- void **handleUnexpectedExceptionNotThrown** ()
- void **handleExceptionNotThrownAsExpected** ()
- void **handleThrowingCallSkipped** ()
- void **handleUnexpectedInflightException** ()
- void **complete** ()
- void **setCompleted** ()
- auto **allowThrows** () const -> bool

5.2.1 Detailed Description

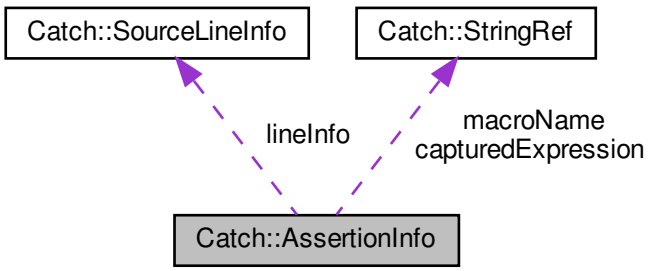
Definition at line 1911 of file catch.hpp.

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

- lib/catch.hpp

5.3 Catch::AssertionInfo Struct Reference

Collaboration diagram for Catch::AssertionInfo:



Public Attributes

- StringRef **macroName**
- SourceLineInfo **lineInfo**
- StringRef **capturedExpression**
- ResultDisposition::Flags **resultDisposition**

5.3.1 Detailed Description

Definition at line 937 of file catch.hpp.

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

- lib/catch.hpp

5.4 Catch::AssertionReaction Struct Reference

Public Attributes

- bool **shouldDebugBreak** = false
- bool **shouldThrow** = false

5.4.1 Detailed Description

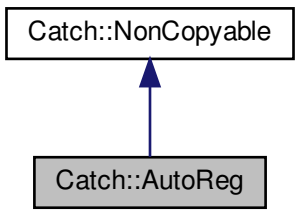
Definition at line 1906 of file catch.hpp.

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

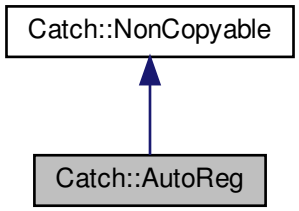
- lib/catch.hpp

5.5 Catch::AutoReg Struct Reference

Inheritance diagram for Catch::AutoReg:



Collaboration diagram for Catch::AutoReg:



Public Member Functions

- **AutoReg** (ITestInvoker *invoker, SourceLineInfo const &lineInfo, StringRef const &classOrMethod, NameAndTags const &nameAndTags) noexcept

5.5.1 Detailed Description

Definition at line 744 of file catch.hpp.

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

- lib/catch.hpp

5.6 Catch::BenchmarkLooper Class Reference

Public Member Functions

- **BenchmarkLooper** (StringRef name)
- **operator bool** ()
- void **increment** ()
- void **reportStart** ()
- auto **needsMoreIterations** () -> bool

5.6.1 Detailed Description

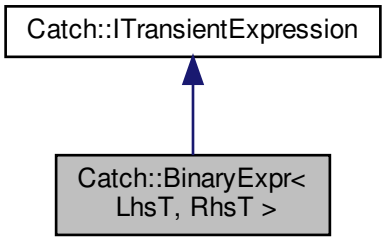
Definition at line 2303 of file catch.hpp.

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

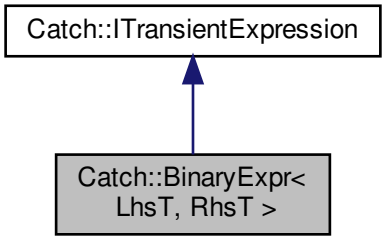
- lib/catch.hpp

5.7 Catch::BinaryExpr< LhsT, RhsT > Class Template Reference

Inheritance diagram for Catch::BinaryExpr< LhsT, RhsT >:



Collaboration diagram for Catch::BinaryExpr< LhsT, RhsT >:



Public Member Functions

- **BinaryExpr** (bool comparisonResult, LhsT lhs, StringRef op, RhsT rhs)

Additional Inherited Members

5.7.1 Detailed Description

template<typename LhsT, typename RhsT>
class Catch::BinaryExpr< LhsT, RhsT >

Definition at line 1681 of file catch.hpp.

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

- lib/catch.hpp

5.8 Catch::Capturer Class Reference

Public Member Functions

- **Capturer** (StringRef macroName, SourceLineInfo const &lineInfo, ResultWas::OfType resultType, StringRef names)
- void **captureValue** (size_t index, std::string const &value)
- template<typename T >
  void **captureValues** (size_t index, T const &value)
- template<typename T , typename... Ts>
  void **captureValues** (size_t index, T const &value, Ts const &... values)

5.8.1 Detailed Description

Definition at line 2012 of file catch.hpp.

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

- lib/catch.hpp

5.9 Catch::Matchers::StdString::CasedString Struct Reference

Public Member Functions

- **CasedString** (std::string const &str, CaseSensitive::Choice caseSensitivity)
- std::string **adjustString** (std::string const &str) const
- std::string **caseSensitivitySuffix** () const

Public Attributes

- CaseSensitive::Choice **m_caseSensitivity**
- std::string **m_str**

5.9.1 Detailed Description

Definition at line 2867 of file catch.hpp.

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

- lib/catch.hpp

5.10 Catch::CaseSensitive Struct Reference

Public Types

- enum **Choice** { **Yes**, **No** }

5.10.1 Detailed Description

Definition at line 386 of file catch.hpp.

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

- lib/catch.hpp

5.11 Catch_global_namespace_dummy Struct Reference

5.11.1 Detailed Description

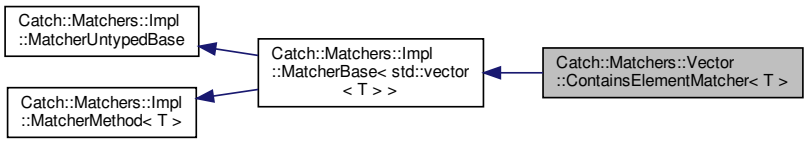
Definition at line 381 of file catch.hpp.

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

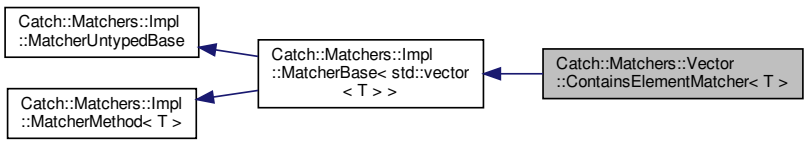
- lib/catch.hpp

5.12 Catch::Matchers::Vector::ContainsElementMatcher< T > Struct Template Reference

Inheritance diagram for Catch::Matchers::Vector::ContainsElementMatcher< T >:



Collaboration diagram for Catch::Matchers::Vector::ContainsElementMatcher< T >:



Public Member Functions

- **ContainsElementMatcher** (T const &comparator)
- bool **match** (std::vector< T > const &v) const override
- std::string **describe** () const override

Public Attributes

- T const & **m_comparator**

Additional Inherited Members

5.12.1 Detailed Description

template<typename T>
struct Catch::Matchers::Vector::ContainsElementMatcher< T >

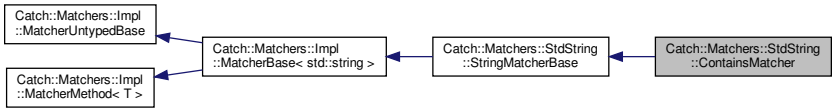
Definition at line 2958 of file catch.hpp.

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

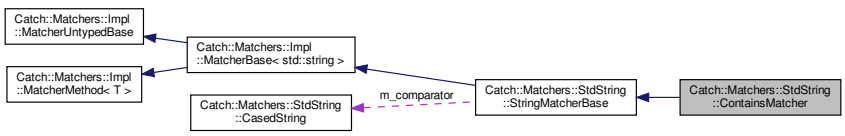
- lib/catch.hpp

5.13 Catch::Matchers::StdString::ContainsMatcher Struct Reference

Inheritance diagram for Catch::Matchers::StdString::ContainsMatcher:



Collaboration diagram for Catch::Matchers::StdString::ContainsMatcher:



Public Member Functions

- **ContainsMatcher** (CasedString const &comparator)
- bool **match** (std::string const &source) const override

Additional Inherited Members

5.13.1 Detailed Description

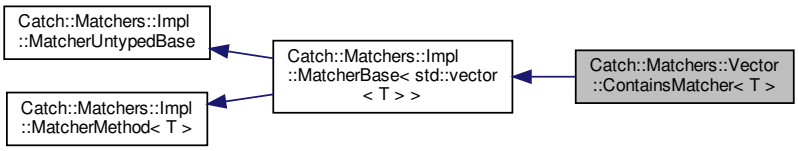
Definition at line 2889 of file catch.hpp.

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

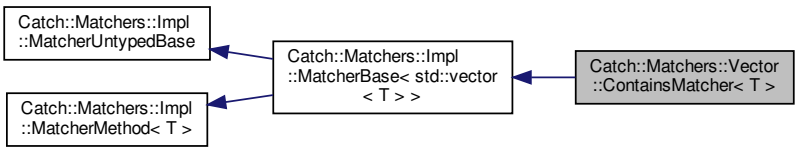
- lib/catch.hpp

5.14 Catch::Matchers::Vector::ContainsMatcher< T > Struct Template Reference

Inheritance diagram for Catch::Matchers::Vector::ContainsMatcher< T >:



Collaboration diagram for Catch::Matchers::Vector::ContainsMatcher< T >:



Public Member Functions

- **ContainsMatcher** (std::vector< T > const &comparator)
- bool **match** (std::vector< T > const &v) const override
- std::string **describe** () const override

Public Attributes

- std::vector< T > const & **m_comparator**

Additional Inherited Members

5.14.1 Detailed Description

template<typename T>
struct Catch::Matchers::Vector::ContainsMatcher< T >

Definition at line 2979 of file catch.hpp.

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

- lib/catch.hpp

5.15 Catch::Counts Struct Reference

Public Member Functions

- Counts **operator-** (Counts const &other) const
- Counts & **operator+=** (Counts const &other)
- std::size_t **total** () const
- bool **allPassed** () const
- bool **allOk** () const

Public Attributes

- std::size_t **passed** = 0
- std::size_t **failed** = 0
- std::size_t **failedButOk** = 0

5.15.1 Detailed Description

Definition at line 2184 of file catch.hpp.

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

- lib/catch.hpp

5.16 Catch::Decomposer Struct Reference

Public Member Functions

- template<typename T >
  auto **operator<=** (T const &lhs) -> ExprLhs< T const &>
- auto **operator<=** (bool value) -> ExprLhs< bool >

5.16.1 Detailed Description

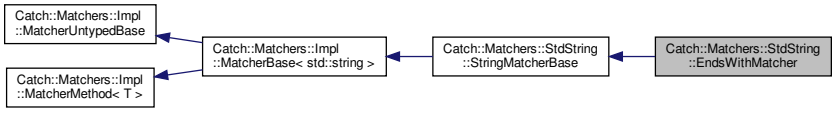
Definition at line 1789 of file catch.hpp.

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

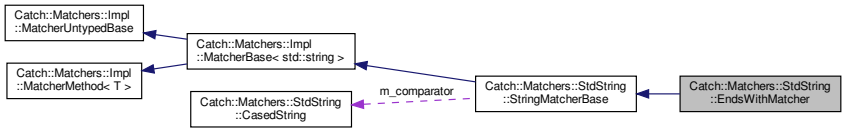
- lib/catch.hpp

5.17 Catch::Matchers::StdString::EndsWithMatcher Struct Reference

Inheritance diagram for Catch::Matchers::StdString::EndsWithMatcher:



Collaboration diagram for Catch::Matchers::StdString::EndsWithMatcher:



Public Member Functions

- **EndsWithMatcher** (CasedString const &comparator)
- bool **match** (std::string const &source) const override

Additional Inherited Members

5.17.1 Detailed Description

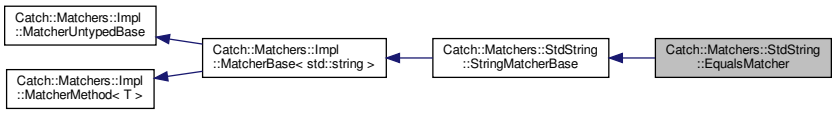
Definition at line 2897 of file catch.hpp.

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

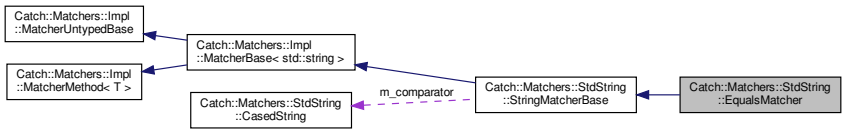
- lib/catch.hpp

5.18 Catch::Matchers::StdString::EqualsMatcher Struct Reference

Inheritance diagram for Catch::Matchers::StdString::EqualsMatcher:



Collaboration diagram for Catch::Matchers::StdString::EqualsMatcher:



Public Member Functions

- **EqualsMatcher** (CasedString const &comparator)
- bool **match** (std::string const &source) const override

Additional Inherited Members

5.18.1 Detailed Description

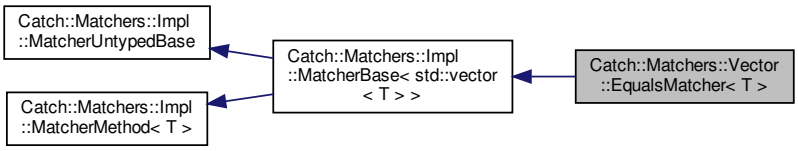
Definition at line 2885 of file catch.hpp.

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

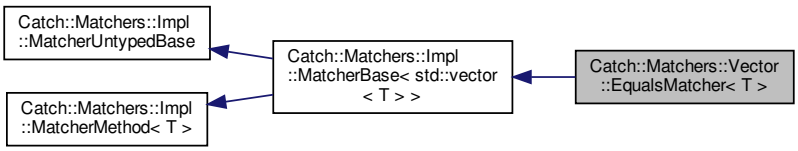
- lib/catch.hpp

5.19 Catch::Matchers::Vector::EqualsMatcher< T > Struct Template Reference

Inheritance diagram for Catch::Matchers::Vector::EqualsMatcher< T >:



Collaboration diagram for Catch::Matchers::Vector::EqualsMatcher< T >:



Public Member Functions

- **EqualsMatcher** (std::vector< T > const &comparator)
- bool **match** (std::vector< T > const &v) const override
- std::string **describe** () const override

Public Attributes

- std::vector< T > const & **m_comparator**

Additional Inherited Members

5.19.1 Detailed Description

template<typename T>
struct Catch::Matchers::Vector::EqualsMatcher< T >

Definition at line 3009 of file catch.hpp.

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

- lib/catch.hpp

5.20 Catch::ExceptionTranslatorRegistrar Class Reference

Public Member Functions

- template<typename T >
  ExceptionTranslatorRegistrar (std::string(*translateFunction)(T &))

5.20.1 Detailed Description

Definition at line 2418 of file catch.hpp.

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

- lib/catch.hpp

5.21 Catch::ExprLhs< LhsT > Class Template Reference

Public Member Functions

- **ExprLhs** (LhsT lhs)
- template<typename RhsT >
  auto **operator==** (RhsT const &rhs) -> BinaryExpr< LhsT, RhsT const &> const
- auto **operator==** (bool rhs) -> BinaryExpr< LhsT, bool > const
- template<typename RhsT >
  auto **operator!=** (RhsT const &rhs) -> BinaryExpr< LhsT, RhsT const &> const
- auto **operator!=** (bool rhs) -> BinaryExpr< LhsT, bool > const
- template<typename RhsT >
  auto **operator>** (RhsT const &rhs) -> BinaryExpr< LhsT, RhsT const &> const
- template<typename RhsT >
  auto **operator<** (RhsT const &rhs) -> BinaryExpr< LhsT, RhsT const &> const
- template<typename RhsT >
  auto **operator>=** (RhsT const &rhs) -> BinaryExpr< LhsT, RhsT const &> const
- template<typename RhsT >
  auto **operator<=** (RhsT const &rhs) -> BinaryExpr< LhsT, RhsT const &> const
- auto **makeUnaryExpr** () const -> UnaryExpr< LhsT >

5.21.1 Detailed Description

template<typename LhsT>
class Catch::ExprLhs< LhsT >

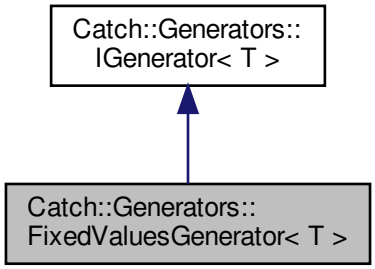
Definition at line 1739 of file catch.hpp.

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

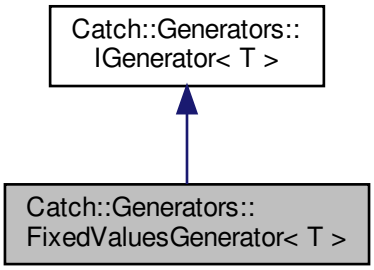
- lib/catch.hpp

5.22 Catch::Generators::FixedValuesGenerator< T > Class Template Reference

Inheritance diagram for Catch::Generators::FixedValuesGenerator< T >:



Collaboration diagram for Catch::Generators::FixedValuesGenerator< T >:



Public Member Functions

- **FixedValuesGenerator** (std::initializer_list< T > values)
- auto **get** (size_t index) const -> T override

5.22.1 Detailed Description

template<typename T>
class Catch::Generators::FixedValuesGenerator< T >

Definition at line 3266 of file catch.hpp.

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

- lib/catch.hpp

5.23 Catch::Generators::Generator< T > Class Template Reference

Public Member Functions

- **Generator** (size_t size, std::unique_ptr< IGenerator< T >> generator)
- auto **size** () const -> size_t
- auto **operator[ ]** (size_t index) const -> T

5.23.1 Detailed Description

template<typename T>
class Catch::Generators::Generator< T >

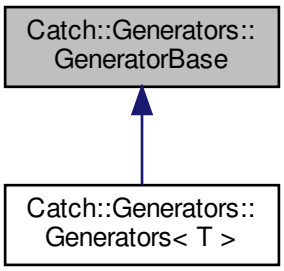
Definition at line 3301 of file catch.hpp.

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

- lib/catch.hpp

5.24 Catch::Generators::GeneratorBase Class Reference

Inheritance diagram for Catch::Generators::GeneratorBase:



Public Member Functions

- **GeneratorBase** (size_t size)
- auto **size** () const -> size_t

Protected Attributes

- size_t **m_size** = 0

5.24.1 Detailed Description

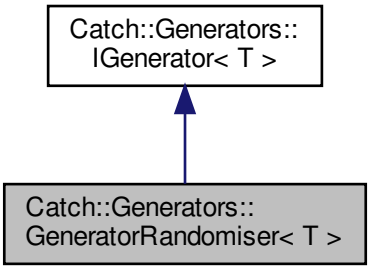
Definition at line 3178 of file catch.hpp.

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

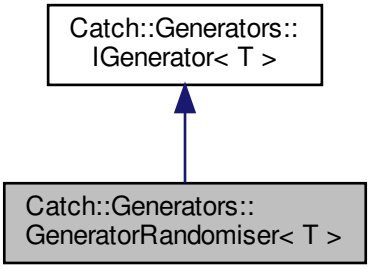
- lib/catch.hpp

5.25 Catch::Generators::GeneratorRandomiser< T > Class Template Reference

Inheritance diagram for Catch::Generators::GeneratorRandomiser< T >:



Collaboration diagram for Catch::Generators::GeneratorRandomiser< T >:



Public Member Functions

- **GeneratorRandomiser** (Generator< T > &&baseGenerator, size_t numberOfItems)
- auto **get** (size_t index) const -> T override

5.25.1 Detailed Description

template<typename T>
class Catch::Generators::GeneratorRandomiser< T >

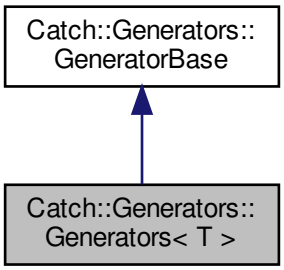
Definition at line 3321 of file catch.hpp.

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

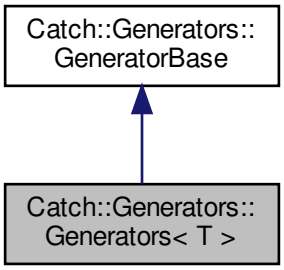
- lib/catch.hpp

5.26 Catch::Generators::Generators< T > Struct Template Reference

Inheritance diagram for Catch::Generators::Generators< T >:



Collaboration diagram for Catch::Generators::Generators< T >:



Public Types

- using **type** = T

Public Member Functions

- void **populate** (T &&val)
- template<typename U >
  void **populate** (U &&val)
- void **populate** (Generator< T > &&generator)
- template<typename U , typename... Gs>
  void **populate** (U &&valueOrGenerator, Gs... moreGenerators)
- auto **operator[ ]** (size_t index) const -> T

Public Attributes

- std::vector< Generator< T > > **m_generators**

Additional Inherited Members

5.26.1 Detailed Description

template<typename T>
struct Catch::Generators::Generators< T >

Definition at line 3382 of file catch.hpp.

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

- lib/catch.hpp

5.27 Catch::IExceptionTranslator Struct Reference

Public Member Functions

- virtual std::string **translate** (ExceptionTranslators::const_iterator it, ExceptionTranslators::const_iterator it↩ End) const =0

5.27.1 Detailed Description

Definition at line 2407 of file catch.hpp.

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

- lib/catch.hpp

5.28 Catch::IExceptionTranslatorRegistry Struct Reference

Public Member Functions

- virtual std::string **translateActiveException** () const =0

5.28.1 Detailed Description

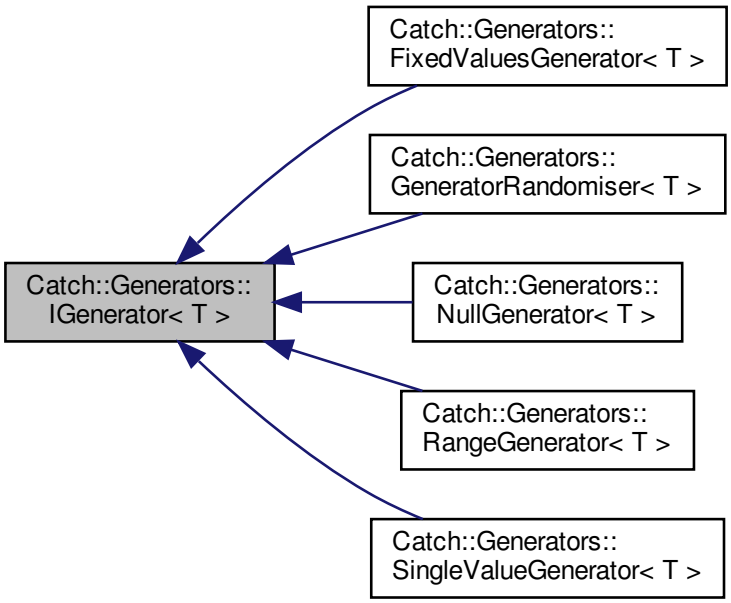
Definition at line 2412 of file catch.hpp.

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

- lib/catch.hpp

5.29 Catch::Generators::IGenerator< T > Struct Template Reference

Inheritance diagram for Catch::Generators::IGenerator< T >:



Public Member Functions

- virtual auto **get** (size_t index) const -> T=0

5.29.1 Detailed Description

template<typename T>
struct Catch::Generators::IGenerator< T >

Definition at line 3249 of file catch.hpp.

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

- lib/catch.hpp

5.30 Catch::IGeneratorTracker Struct Reference

Public Member Functions

- virtual auto **hasGenerator** () const -> bool=0
- virtual auto **getGenerator** () const -> Generators::GeneratorBasePtr const &=0
- virtual void **setGenerator** (Generators::GeneratorBasePtr &&generator)=0
- virtual auto **getIndex** () const -> std::size_t=0

5.30.1 Detailed Description

Definition at line 3191 of file catch.hpp.

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

- lib/catch.hpp

5.31 Catch::IMutableRegistryHub Struct Reference

Public Member Functions

- virtual void **registerReporter** (std::string const &name, IReporterFactoryPtr const &factory)=0
- virtual void **registerListener** (IReporterFactoryPtr const &factory)=0
- virtual void **registerTest** (TestCase const &testInfo)=0
- virtual void **registerTranslator** (const IExceptionTranslator *translator)=0
- virtual void **registerTagAlias** (std::string const &alias, std::string const &tag, SourceLineInfo const &line↩ Info)=0
- virtual void **registerStartupException** () noexcept=0

5.31.1 Detailed Description

Definition at line 2374 of file catch.hpp.

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

- lib/catch.hpp

5.32 Catch::IRegistryHub Struct Reference

Public Member Functions

- virtual IReporterRegistry const & **getReporterRegistry** () const =0
- virtual ITestCaseRegistry const & **getTestCaseRegistry** () const =0
- virtual ITagAliasRegistry const & **getTagAliasRegistry** () const =0
- virtual IExceptionTranslatorRegistry const & **getExceptionTranslatorRegistry** () const =0
- virtual StartupExceptionRegistry const & **getStartupExceptionRegistry** () const =0

5.32.1 Detailed Description

Definition at line 2362 of file catch.hpp.

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

- lib/catch.hpp

5.33 Catch::IResultCapture Struct Reference

Public Member Functions

- virtual bool **sectionStarted** (SectionInfo const &sectionInfo, Counts &assertions)=0
- virtual void **sectionEnded** (SectionEndInfo const &endInfo)=0
- virtual void **sectionEndedEarly** (SectionEndInfo const &endInfo)=0
- virtual auto **acquireGeneratorTracker** (SourceLineInfo const &lineInfo) -> IGeneratorTracker &=0
- virtual void **benchmarkStarting** (BenchmarkInfo const &info)=0
- virtual void **benchmarkEnded** (BenchmarkStats const &stats)=0
- virtual void **pushScopedMessage** (MessageInfo const &message)=0
- virtual void **popScopedMessage** (MessageInfo const &message)=0
- virtual void **handleFatalErrorCondition** (StringRef message)=0
- virtual void **handleExpr** (AssertionInfo const &info, ITransientExpression const &expr, AssertionReaction &reaction)=0
- virtual void **handleMessage** (AssertionInfo const &info, ResultWas::OfType resultType, StringRef const &message, AssertionReaction &reaction)=0
- virtual void **handleUnexpectedExceptionNotThrown** (AssertionInfo const &info, AssertionReaction &reaction)=0
- virtual void **handleUnexpectedInflightException** (AssertionInfo const &info, std::string const &message, AssertionReaction &reaction)=0
- virtual void **handleIncomplete** (AssertionInfo const &info)=0
- virtual void **handleNonExpr** (AssertionInfo const &info, ResultWas::OfType resultType, AssertionReaction &reaction)=0
- virtual bool **lastAssertionPassed** ()=0
- virtual void **assertionPassed** ()=0
- virtual std::string **getCurrentTestName** () const =0
- virtual const AssertionResult * **getLastResult** () const =0
- virtual void **exceptionEarlyReported** ()=0

5.33.1 Detailed Description

Definition at line 1827 of file catch.hpp.

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

- lib/catch.hpp

5.34 Catch::IRunner Struct Reference

Public Member Functions

- virtual bool **aborting** () const =0

5.34.1 Detailed Description

Definition at line 3552 of file catch.hpp.

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

- lib/catch.hpp

5.35 Catch::is_range< T > Struct Template Reference

Static Public Attributes

- static const bool **value**

5.35.1 Detailed Description

template<typename T>
struct Catch::is_range< T >

Definition at line 1474 of file catch.hpp.

5.35.2 Member Data Documentation

5.35.2.1 value

```
template<typename T >
const bool Catch::is_range< T >::value  [static]
```

Initial value:
=
```
            !std::is_same<decltype(begin(std::declval<T>())), not_this_one>::value &&
            !std::is_same<decltype(end(std::declval<T>())), not_this_one>::value
```

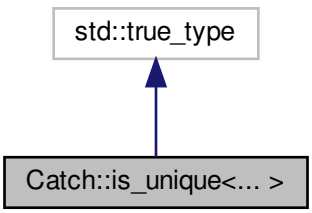
Definition at line 1475 of file catch.hpp.

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

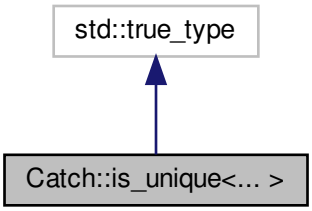
- lib/catch.hpp

5.36 Catch::is_unique<... > Struct Template Reference

Inheritance diagram for Catch::is_unique<... >:



Collaboration diagram for Catch::is_unique<... >:



5.36.1 Detailed Description

template<typename...>
struct Catch::is_unique<... >

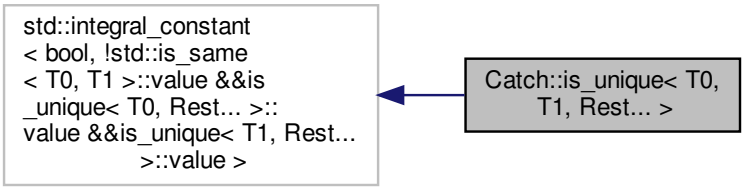
Definition at line 637 of file catch.hpp.

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

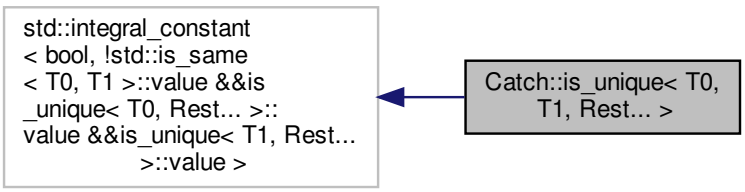
- lib/catch.hpp

5.37 Catch::is_unique< T0, T1, Rest... > Struct Template Reference

Inheritance diagram for Catch::is_unique< T0, T1, Rest... >:



Collaboration diagram for Catch::is_unique< T0, T1, Rest... >:



5.37.1 Detailed Description

template<typename T0, typename T1, typename... Rest>
struct Catch::is_unique< T0, T1, Rest... >

Definition at line 640 of file catch.hpp.

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

- lib/catch.hpp

5.38 Catch::Detail::IsStreamInsertable< T > Class Template Reference

Static Public Attributes

- static const bool **value** = decltype(test<std::ostream, const T&>(0))::value

5.38.1 Detailed Description

template<typename T>
class Catch::Detail::IsStreamInsertable< T >

Definition at line 1069 of file catch.hpp.

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

- lib/catch.hpp

5.39 Catch::IStream Struct Reference

Public Member Functions

- virtual std::ostream & **stream** () const =0

5.39.1 Detailed Description

Definition at line 974 of file catch.hpp.

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

- lib/catch.hpp

5.40 Catch::ITestCaseRegistry Struct Reference

Public Member Functions

- virtual std::vector< TestCase > const & **getAllTests** () const =0
- virtual std::vector< TestCase > const & **getAllTestsSorted** (IConfig const &config) const =0

5.40.1 Detailed Description

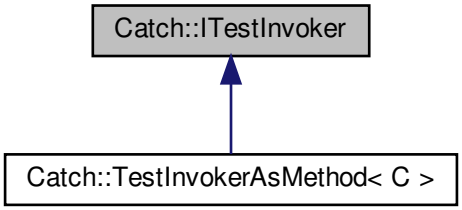
Definition at line 482 of file catch.hpp.

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

- lib/catch.hpp

5.41 Catch::ITestInvoker Struct Reference

Inheritance diagram for Catch::ITestInvoker:



Public Member Functions

- virtual void **invoke** () const =0

5.41.1 Detailed Description

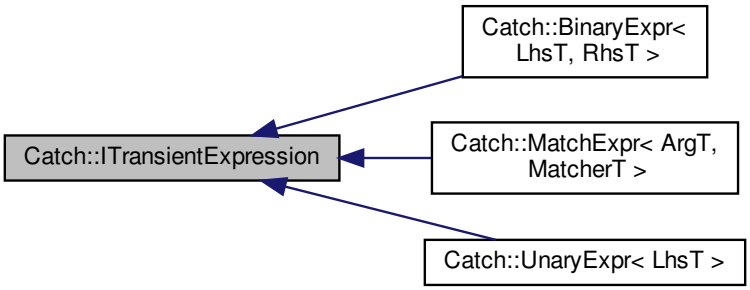
Definition at line 472 of file catch.hpp.

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

- lib/catch.hpp

5.42 Catch::ITransientExpression Struct Reference

Inheritance diagram for Catch::ITransientExpression:



Public Member Functions

- auto **isBinaryExpression** () const -> bool
- auto **getResult** () const -> bool
- virtual void **streamReconstructedExpression** (std::ostream &os) const =0
- **ITransientExpression** (bool isBinaryExpression, bool result)

Public Attributes

- bool **m_isBinaryExpression**
- bool **m_result**

5.42.1 Detailed Description

Definition at line 1659 of file catch.hpp.

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

- lib/catch.hpp

5.43 Catch::LazyExpression Class Reference

Public Member Functions

- **LazyExpression** (bool isNegated)
- **LazyExpression** (LazyExpression const &other)
- LazyExpression & **operator=** (LazyExpression const &)=delete
- **operator bool** () const

Friends

- class **AssertionHandler**
- struct **AssertionStats**
- class **RunContext**
- auto **operator**<< (std::ostream &os, LazyExpression const &lazyExpr) -> std::ostream &

5.43.1 Detailed Description

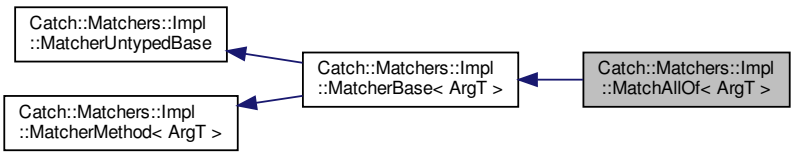
Definition at line 1889 of file catch.hpp.

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

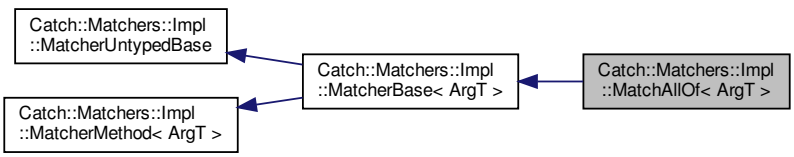
- lib/catch.hpp

5.44 Catch::Matchers::Impl::MatchAllOf< ArgT > Struct Template Reference

Inheritance diagram for Catch::Matchers::Impl::MatchAllOf< ArgT >:



Collaboration diagram for Catch::Matchers::Impl::MatchAllOf< ArgT >:



Public Member Functions

- bool **match** (ArgT const &arg) const override
- std::string **describe** () const override
- MatchAllOf< ArgT > & **operator&&** (MatcherBase< ArgT > const &other)

Public Attributes

- std::vector< MatcherBase< ArgT > const * > **m_matchers**

Additional Inherited Members

5.44.1 Detailed Description

template<typename ArgT>
struct Catch::Matchers::Impl::MatchAllOf< ArgT >

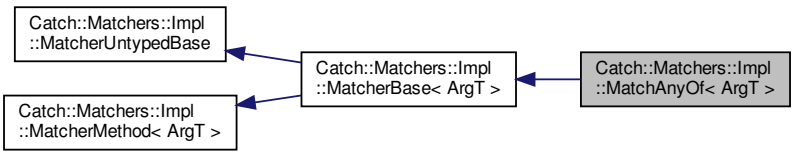
Definition at line 2624 of file catch.hpp.

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

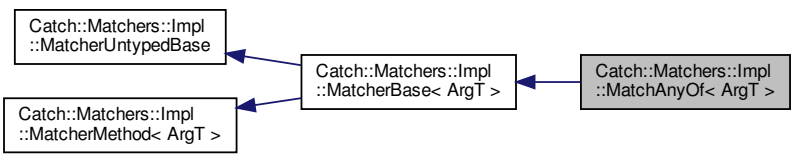
- lib/catch.hpp

5.45 Catch::Matchers::Impl::MatchAnyOf< ArgT > Struct Template Reference

Inheritance diagram for Catch::Matchers::Impl::MatchAnyOf< ArgT >:



Collaboration diagram for Catch::Matchers::Impl::MatchAnyOf< ArgT >:



Public Member Functions

- bool **match** (ArgT const &arg) const override
- std::string **describe** () const override
- MatchAnyOf< ArgT > & **operator||** (MatcherBase< ArgT > const &other)

Public Attributes

- std::vector< MatcherBase< ArgT > const * > **m_matchers**

Additional Inherited Members

5.45.1 Detailed Description

template<typename ArgT>
struct Catch::Matchers::Impl::MatchAnyOf< ArgT >

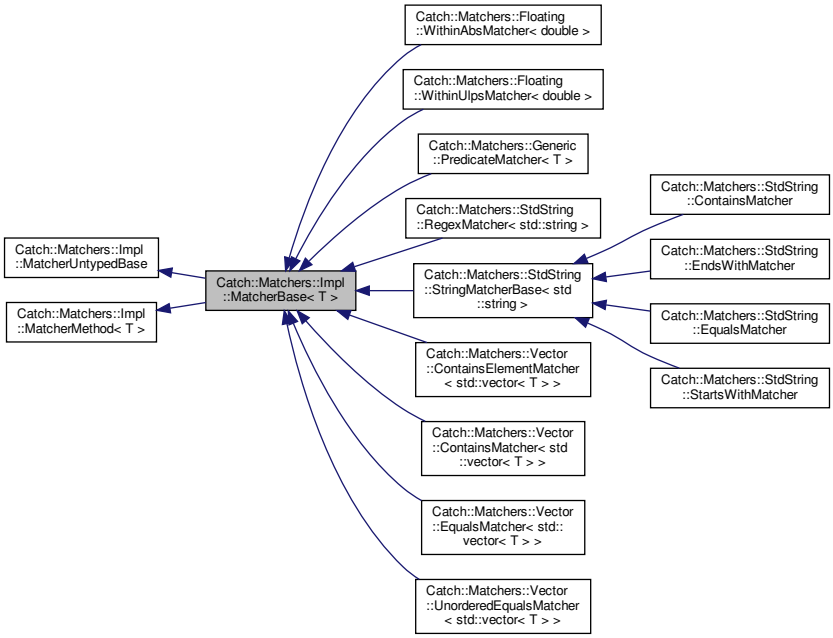
Definition at line 2625 of file catch.hpp.

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

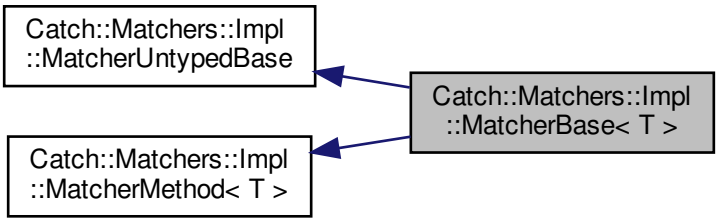
- lib/catch.hpp

5.46 Catch::Matchers::Impl::MatcherBase< T > Struct Template Reference

Inheritance diagram for Catch::Matchers::Impl::MatcherBase< T >:



Collaboration diagram for Catch::Matchers::Impl::MatcherBase< T >:



Public Member Functions

- MatchAllOf< T > **operator&&** (MatcherBase const &other) const
- MatchAnyOf< T > **operator||** (MatcherBase const &other) const
- MatchNotOf< T > **operator!** () const

Additional Inherited Members

5.46.1 Detailed Description

template<typename T>
struct Catch::Matchers::Impl::MatcherBase< T >

Definition at line 2656 of file catch.hpp.

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

- lib/catch.hpp

5.47 Catch::Matchers::Impl::MatcherMethod< ObjectT > Struct Template Reference

Public Member Functions

- virtual bool **match** (ObjectT const &arg) const =0

5.47.1 Detailed Description

template<typename ObjectT>
struct Catch::Matchers::Impl::MatcherMethod< ObjectT >

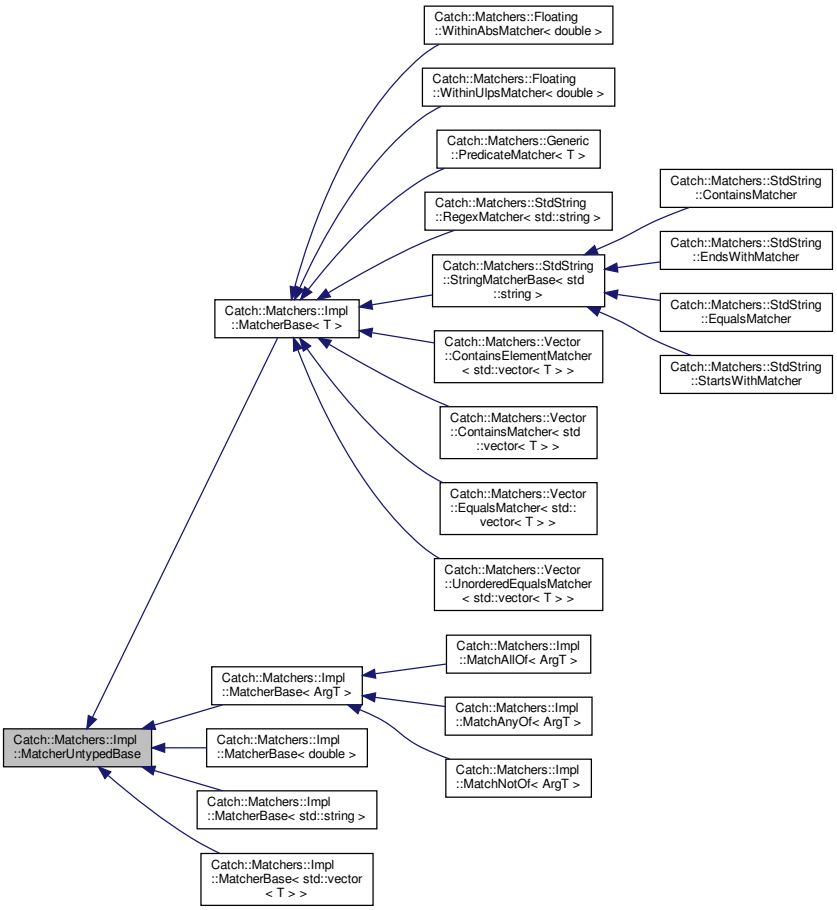
Definition at line 2647 of file catch.hpp.

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

- lib/catch.hpp

5.48 Catch::Matchers::Impl::MatcherUntypedBase Class Reference

Inheritance diagram for Catch::Matchers::Impl::MatcherUntypedBase:



Public Member Functions

- **MatcherUntypedBase** (MatcherUntypedBase const &)=default
- MatcherUntypedBase & **operator=** (MatcherUntypedBase const &)=delete
- std::string **toString** () const

Protected Member Functions

- virtual std::string **describe** () const =0

Protected Attributes

- std::string **m_cachedToString**

5.48.1 Detailed Description

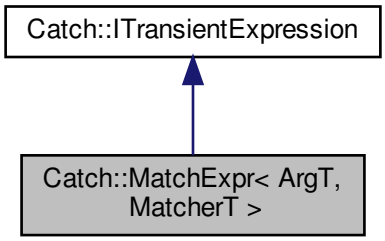
Definition at line 2628 of file catch.hpp.

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

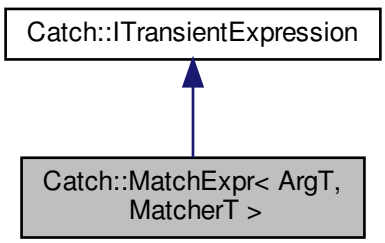
- lib/catch.hpp

5.49 Catch::MatchExpr< ArgT, MatcherT > Class Template Reference

Inheritance diagram for Catch::MatchExpr< ArgT, MatcherT >:



Collaboration diagram for Catch::MatchExpr< ArgT, MatcherT >:



Public Member Functions

- **MatchExpr** (ArgT const &arg, MatcherT const &matcher, StringRef const &matcherString)
- void **streamReconstructedExpression** (std::ostream &os) const override

Additional Inherited Members

5.49.1 Detailed Description

template<typename ArgT, typename MatcherT>
class Catch::MatchExpr< ArgT, MatcherT >

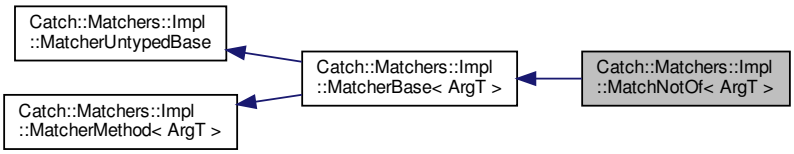
Definition at line 3103 of file catch.hpp.

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

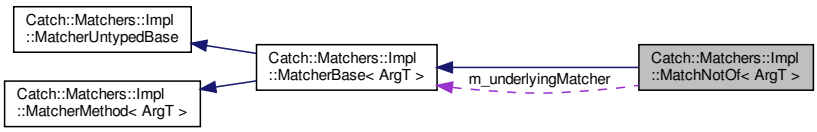
- lib/catch.hpp

5.50 Catch::Matchers::Impl::MatchNotOf< ArgT > Struct Template Reference

Inheritance diagram for Catch::Matchers::Impl::MatchNotOf< ArgT >:



Collaboration diagram for Catch::Matchers::Impl::MatchNotOf< ArgT >:



Public Member Functions

- **MatchNotOf** (MatcherBase< ArgT > const &underlyingMatcher)
- bool **match** (ArgT const &arg) const override
- std::string **describe** () const override

Public Attributes

- MatcherBase< ArgT > const & **m_underlyingMatcher**

Additional Inherited Members

5.50.1 Detailed Description

template<typename ArgT>
struct Catch::Matchers::Impl::MatchNotOf< ArgT >

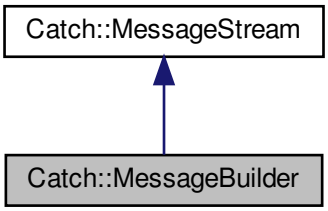
Definition at line 2626 of file catch.hpp.

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

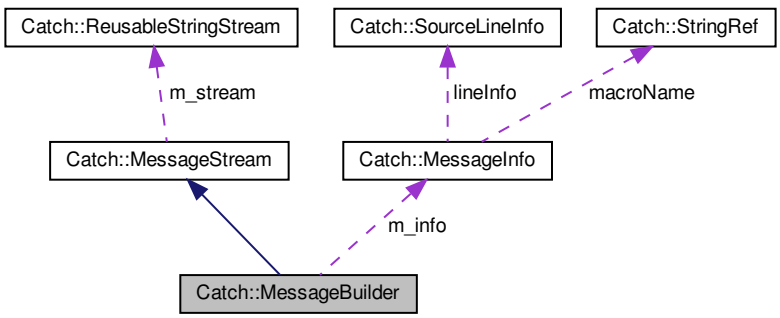
- lib/catch.hpp

5.51 Catch::MessageBuilder Struct Reference

Inheritance diagram for Catch::MessageBuilder:



Collaboration diagram for Catch::MessageBuilder:



Public Member Functions

- **MessageBuilder** (StringRef const &macroName, SourceLineInfo const &lineInfo, ResultWas::OfType type)
- template<typename T >
  MessageBuilder & **operator**<< (T const &value)

Public Attributes

- MessageInfo **m_info**

5.51.1 Detailed Description

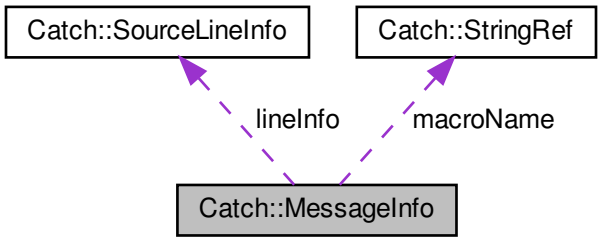
Definition at line 1990 of file catch.hpp.

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

- lib/catch.hpp

5.52 Catch::MessageInfo Struct Reference

Collaboration diagram for Catch::MessageInfo:



Public Member Functions

- **MessageInfo** (StringRef const &_macroName, SourceLineInfo const &_lineInfo, ResultWas::OfType _type)
- bool **operator==** (MessageInfo const &other) const
- bool **operator**< (MessageInfo const &other) const

Public Attributes

- StringRef **macroName**
- std::string **message**
- SourceLineInfo **lineInfo**
- ResultWas::OfType **type**
- unsigned int **sequence**

5.52.1 Detailed Description

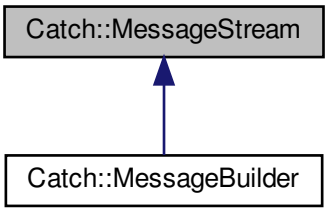
Definition at line 1962 of file catch.hpp.

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

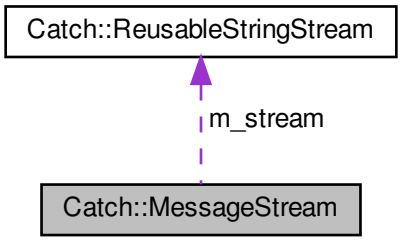
- lib/catch.hpp

5.53 Catch::MessageStream Struct Reference

Inheritance diagram for Catch::MessageStream:



Collaboration diagram for Catch::MessageStream:



Public Member Functions

- template<typename T >
  MessageStream & **operator**<< (T const &value)

Public Attributes

- ReusableStringStream **m_stream**

5.53.1 Detailed Description

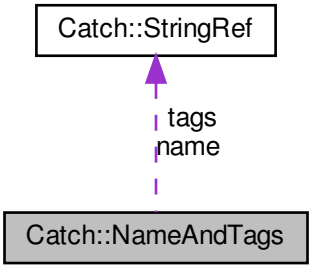
Definition at line 1979 of file catch.hpp.

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

- lib/catch.hpp

5.54 Catch::NameAndTags Struct Reference

Collaboration diagram for Catch::NameAndTags:



Public Member Functions

- **NameAndTags** (StringRef const &name_=StringRef(), StringRef const &tags_=StringRef()) noexcept

Public Attributes

- StringRef **name**
- StringRef **tags**

5.54.1 Detailed Description

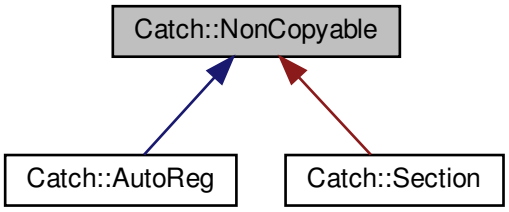
Definition at line 738 of file catch.hpp.

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

- lib/catch.hpp

5.55 Catch::NonCopyable Class Reference

Inheritance diagram for Catch::NonCopyable:



5.55.1 Detailed Description

Definition at line 391 of file catch.hpp.

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

- lib/catch.hpp

5.56 Catch::not_this_one Struct Reference

5.56.1 Detailed Description

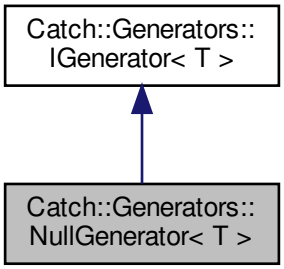
Definition at line 1464 of file catch.hpp.

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

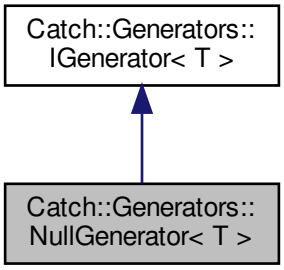
- lib/catch.hpp

5.57 Catch::Generators::NullGenerator< T > Struct Template Reference

Inheritance diagram for Catch::Generators::NullGenerator< T >:



Collaboration diagram for Catch::Generators::NullGenerator< T >:



Public Member Functions

- auto **get** (size_t) const -> T override

5.57.1 Detailed Description

template<typename T>
struct Catch::Generators::NullGenerator< T >

Definition at line 3294 of file catch.hpp.

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

- lib/catch.hpp

5.58 Catch::pluralise Struct Reference

Public Member Functions

- **pluralise** (std::size_t count, std::string const &label)

Public Attributes

- std::size_t **m_count**
- std::string **m_label**

Friends

- std::ostream & **operator<<** (std::ostream &os, pluralise const &pluraliser)

5.58.1 Detailed Description

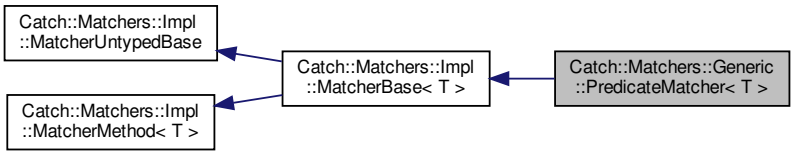
Definition at line 2601 of file catch.hpp.

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

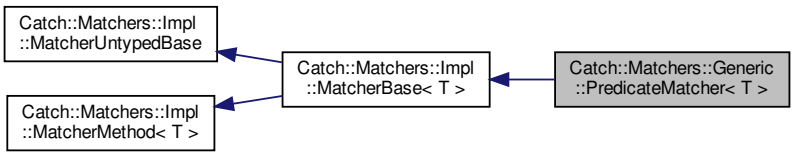
- lib/catch.hpp

5.59 Catch::Matchers::Generic::PredicateMatcher< T > Class Template Reference

Inheritance diagram for Catch::Matchers::Generic::PredicateMatcher< T >:



Collaboration diagram for Catch::Matchers::Generic::PredicateMatcher< T >:



Public Member Functions

- **PredicateMatcher** (std::function< bool(T const &)> const &elem, std::string const &descr)
- bool **match** (T const &item) const override
- std::string **describe** () const override

Additional Inherited Members

5.59.1 Detailed Description

template<typename T>
class Catch::Matchers::Generic::PredicateMatcher< T >

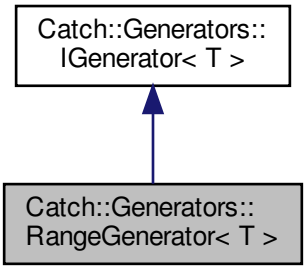
Definition at line 2824 of file catch.hpp.

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

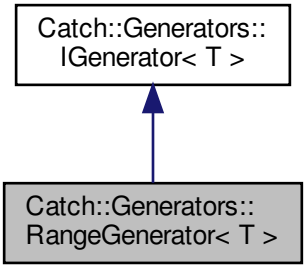
- lib/catch.hpp

5.60 Catch::Generators::RangeGenerator< T > Class Template Reference

Inheritance diagram for Catch::Generators::RangeGenerator< T >:



Collaboration diagram for Catch::Generators::RangeGenerator< T >:



Public Member Functions

- **RangeGenerator** (T const &first, T const &last)
- auto **get** (size_t index) const -> T override

5.60.1 Detailed Description

template<typename T>
class Catch::Generators::RangeGenerator< T >

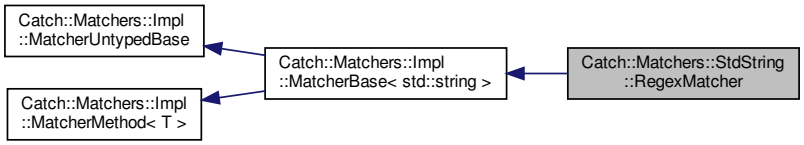
Definition at line 3278 of file catch.hpp.

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

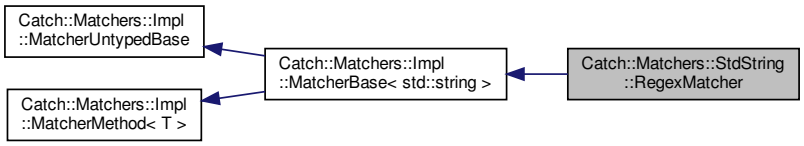
- lib/catch.hpp

5.61 Catch::Matchers::StdString::RegexMatcher Struct Reference

Inheritance diagram for Catch::Matchers::StdString::RegexMatcher:



Collaboration diagram for Catch::Matchers::StdString::RegexMatcher:



Public Member Functions

- **RegexMatcher** (std::string regex, CaseSensitive::Choice caseSensitivity)
- bool **match** (std::string const &matchee) const override
- std::string **describe** () const override

Additional Inherited Members

5.61.1 Detailed Description

Definition at line 2902 of file catch.hpp.

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

- lib/catch.hpp

5.62 Catch::RegistrarForTagAliases Struct Reference

Public Member Functions

- **RegistrarForTagAliases** (char const *alias, char const *tag, SourceLineInfo const &lineInfo)

5.62.1 Detailed Description

Definition at line 449 of file catch.hpp.

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

- lib/catch.hpp

5.63 Catch::Generators::RequiresASpecialisationFor< T > Struct Template Reference

5.63.1 Detailed Description

template<typename T>
struct Catch::Generators::RequiresASpecialisationFor< T >

Definition at line 3337 of file catch.hpp.

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

- lib/catch.hpp

5.64 Catch::ResultDisposition Struct Reference

Public Types

- enum **Flags** { **Normal** = 0x01, **ContinueOnFailure** = 0x02, **FalseTest** = 0x04, **SuppressFail** = 0x08 }

5.64.1 Detailed Description

Definition at line 918 of file catch.hpp.

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

- lib/catch.hpp

5.65 Catch::ResultWas Struct Reference

Public Types

- enum **OfType** {
  Unknown = -1, **Ok** = 0, **Info** = 1, **Warning** = 2,
  FailureBit = 0x10, **ExpressionFailed** = FailureBit | 1, **ExplicitFailure** = FailureBit | 2, **Exception** = 0x100 | FailureBit,
  ThrewException = Exception | 1, **DidntThrowException** = Exception | 2, **FatalErrorCondition** = 0x200 | FailureBit }

5.65.1 Detailed Description

Definition at line 894 of file catch.hpp.

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

- lib/catch.hpp

5.66 Catch::ReusableStringStream Class Reference

Public Member Functions

- auto **str** () const -> std::string
- template<typename T >
  auto **operator**<< (T const &value) -> ReusableStringStream &
- auto **get** () -> std::ostream &

5.66.1 Detailed Description

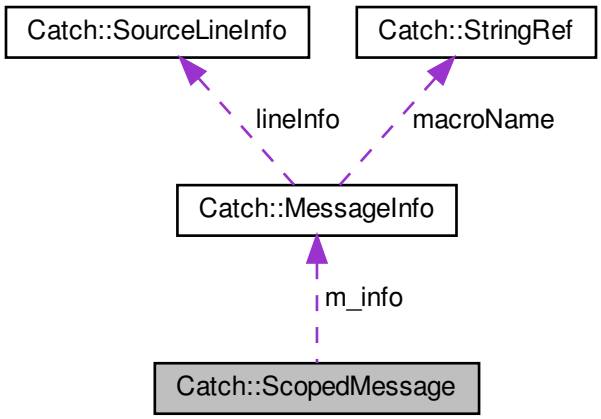
Definition at line 981 of file catch.hpp.

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

- lib/catch.hpp

5.67 Catch::ScopedMessage Class Reference

Collaboration diagram for Catch::ScopedMessage:



Public Member Functions

- **ScopedMessage** (MessageBuilder const &builder)

Public Attributes

- MessageInfo **m_info**

5.67.1 Detailed Description

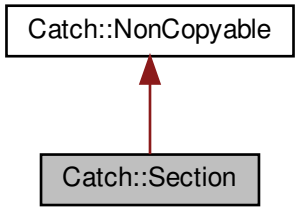
Definition at line 2004 of file catch.hpp.

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

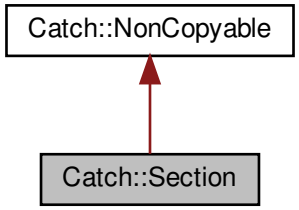
- lib/catch.hpp

5.68 Catch::Section Class Reference

Inheritance diagram for Catch::Section:



Collaboration diagram for Catch::Section:



Public Member Functions

- **Section** (SectionInfo const &info)
- **operator bool** () const

5.68.1 Detailed Description

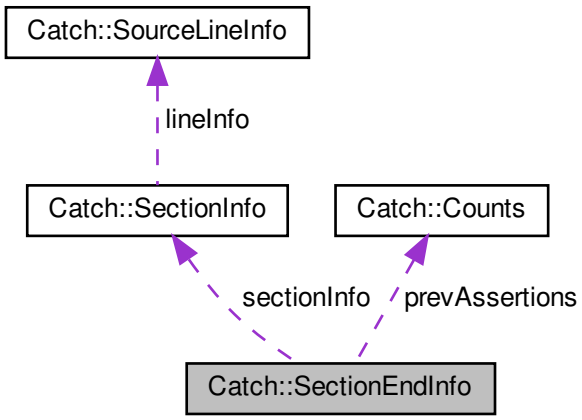
Definition at line 2266 of file catch.hpp.

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

- lib/catch.hpp

5.69 Catch::SectionEndInfo Struct Reference

Collaboration diagram for Catch::SectionEndInfo:



Public Attributes

- SectionInfo **sectionInfo**
- Counts **prevAssertions**
- double **durationInSeconds**

5.69.1 Detailed Description

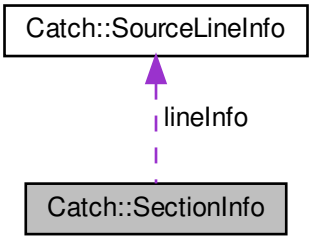
Definition at line 2231 of file catch.hpp.

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

- lib/catch.hpp

5.70 Catch::SectionInfo Struct Reference

Collaboration diagram for Catch::SectionInfo:



Public Member Functions

- **SectionInfo** (SourceLineInfo const &_lineInfo, std::string const &_name)
- **SectionInfo** (SourceLineInfo const &_lineInfo, std::string const &_name, std::string const &)

Public Attributes

- std::string **name**
- std::string **description**
- SourceLineInfo **lineInfo**

5.70.1 Detailed Description

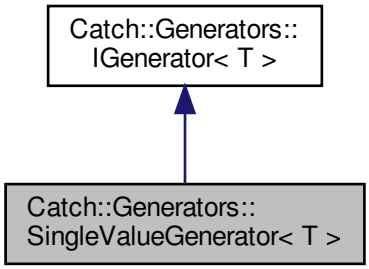
Definition at line 2215 of file catch.hpp.

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

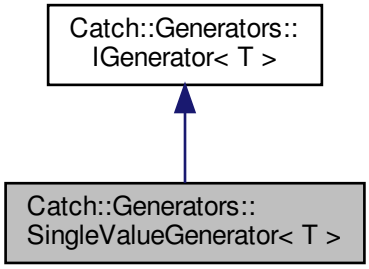
- lib/catch.hpp

5.71 Catch::Generators::SingleValueGenerator< T > Class Template Reference

Inheritance diagram for Catch::Generators::SingleValueGenerator< T >:



Collaboration diagram for Catch::Generators::SingleValueGenerator< T >:



Public Member Functions

- **SingleValueGenerator** (T const &value)
- auto **get** (size_t) const -> T override

5.71.1 Detailed Description

template<typename T>
class Catch::Generators::SingleValueGenerator< T >

Definition at line 3255 of file catch.hpp.

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

- lib/catch.hpp

5.72 Catch::SourceLineInfo Struct Reference

Public Member Functions

- **SourceLineInfo** (char const *_file, std::size_t _line) noexcept
- **SourceLineInfo** (SourceLineInfo const &other)=default
- **SourceLineInfo** (SourceLineInfo &&)=default
- SourceLineInfo & **operator=** (SourceLineInfo const &)=default
- SourceLineInfo & **operator=** (SourceLineInfo &&)=default
- bool **empty** () const noexcept
- bool **operator==** (SourceLineInfo const &other) const noexcept
- bool **operator<** (SourceLineInfo const &other) const noexcept

Public Attributes

- char const * **file**
- std::size_t **line**

5.72.1 Detailed Description

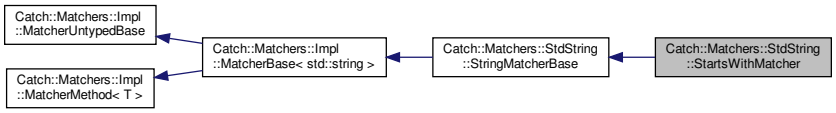
Definition at line 402 of file catch.hpp.

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

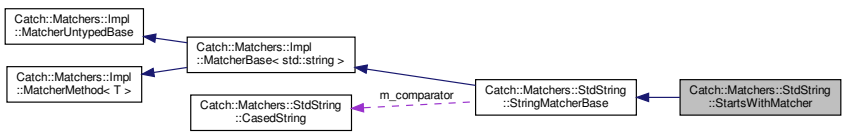
- lib/catch.hpp

5.73 Catch::Matchers::StdString::StartsWithMatcher Struct Reference

Inheritance diagram for Catch::Matchers::StdString::StartsWithMatcher:



Collaboration diagram for Catch::Matchers::StdString::StartsWithMatcher:



Public Member Functions

- **StartsWithMatcher** (CasedString const &comparator)
- bool **match** (std::string const &source) const override

Additional Inherited Members

5.73.1 Detailed Description

Definition at line 2893 of file catch.hpp.

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

- lib/catch.hpp

5.74 Catch::StreamEndStop Struct Reference

Public Member Functions

- std::string **operator+** () const

5.74.1 Detailed Description

Definition at line 434 of file catch.hpp.

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

- lib/catch.hpp

5.75 Catch::StringMaker< T, typename > Struct Template Reference

Static Public Member Functions

- template<typename Fake = T>
  static std::enable_if<::Catch::Detail::IsStreamInsertable< Fake >::value, std::string >::type **convert** (const Fake &value)
- template<typename Fake = T>
  static std::enable_if<!::Catch::Detail::IsStreamInsertable< Fake >::value, std::string >::type **convert** (const Fake &value)

5.75.1 Detailed Description

template<typename T, typename = void>
struct Catch::StringMaker< T, typename >

Definition at line 1120 of file catch.hpp.

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

- lib/catch.hpp

5.76 Catch::StringMaker< bool > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (bool b)

5.76.1 Detailed Description

template<>
struct Catch::StringMaker< bool >

Definition at line 1260 of file catch.hpp.

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

- lib/catch.hpp

5.77 Catch::StringMaker< Catch::Detail::Approx > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (Catch::Detail::Approx const &value)

5.77.1 Detailed Description

template<>
struct Catch::StringMaker< Catch::Detail::Approx >

Definition at line 2577 of file catch.hpp.

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

- lib/catch.hpp

5.78 Catch::StringMaker< char * > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (char *str)

5.78.1 Detailed Description

template<>
struct Catch::StringMaker< char * >

Definition at line 1186 of file catch.hpp.

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

- lib/catch.hpp

5.79 Catch::StringMaker< char > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (char c)

5.79.1 Detailed Description

template<>
struct Catch::StringMaker< char >

Definition at line 1265 of file catch.hpp.

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

- lib/catch.hpp

5.80 Catch::StringMaker< char const * > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (char const *str)

5.80.1 Detailed Description

template<>
struct Catch::StringMaker< char const * >

Definition at line 1182 of file catch.hpp.

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

- lib/catch.hpp

5.81 Catch::StringMaker< char[SZ]> Struct Template Reference

Static Public Member Functions

- static std::string **convert** (char const *str)

5.81.1 Detailed Description

template<int SZ>
struct Catch::StringMaker< char[SZ]>

Definition at line 1216 of file catch.hpp.

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

- lib/catch.hpp

5.82 Catch::StringMaker< double > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (double value)

5.82.1 Detailed Description

template<>
struct Catch::StringMaker< double >

Definition at line 1287 of file catch.hpp.

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

- lib/catch.hpp

5.83 Catch::StringMaker< float > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (float value)

5.83.1 Detailed Description

template<>
struct Catch::StringMaker< float >

Definition at line 1283 of file catch.hpp.

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

- lib/catch.hpp

5.84 Catch::StringMaker< int > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (int value)

5.84.1 Detailed Description

template<>
struct Catch::StringMaker< int >

Definition at line 1235 of file catch.hpp.

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

- lib/catch.hpp

5.85 Catch::StringMaker< long > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (long value)

5.85.1 Detailed Description

template<>
struct Catch::StringMaker< long >

Definition at line 1239 of file catch.hpp.

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

- lib/catch.hpp

5.86 Catch::StringMaker< long long > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (long long value)

5.86.1 Detailed Description

template<>
struct Catch::StringMaker< long long >

Definition at line 1243 of file catch.hpp.

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

- lib/catch.hpp

5.87 Catch::StringMaker< R C::* > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (R C::*p)

5.87.1 Detailed Description

template<typename R, typename C>
struct Catch::StringMaker< R C::* >

Definition at line 1304 of file catch.hpp.

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

- lib/catch.hpp

5.88 Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::IsStreamInsertable< R >::value >::type > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (R const &range)

5.88.1 Detailed Description

template<typename R>
struct Catch::StringMaker< R, typename std::enable_if< is_range< R >::value &&!::Catch::Detail::IsStreamInsertable< R >::value >::type >

Definition at line 1510 of file catch.hpp.

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

- lib/catch.hpp

5.89 Catch::StringMaker< signed char > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (signed char c)

5.89.1 Detailed Description

template<>
struct Catch::StringMaker< signed char >

Definition at line 1269 of file catch.hpp.

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

- lib/catch.hpp

5.90 Catch::StringMaker< signed char[SZ]> Struct Template Reference

Static Public Member Functions

- static std::string **convert** (signed char const *str)

5.90.1 Detailed Description

template<int SZ>
struct Catch::StringMaker< signed char[SZ]>

Definition at line 1222 of file catch.hpp.

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

- lib/catch.hpp

5.91 Catch::StringMaker< std::nullptr_t > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (std::nullptr_t)

5.91.1 Detailed Description

template<>
struct Catch::StringMaker< std::nullptr_t >

Definition at line 1278 of file catch.hpp.

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

- lib/catch.hpp

5.92 Catch::StringMaker< std::string > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (const std::string &str)

5.92.1 Detailed Description

template<>
struct Catch::StringMaker< std::string >

Definition at line 1170 of file catch.hpp.

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

- lib/catch.hpp

5.93 Catch::StringMaker< std::wstring > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (const std::wstring &wstr)

5.93.1 Detailed Description

template<>
struct Catch::StringMaker< std::wstring >

Definition at line 1192 of file catch.hpp.

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

- lib/catch.hpp

5.94 Catch::StringMaker< T * > Struct Template Reference

Static Public Member Functions

- template<typename U >
  static std::string **convert** (U *p)

5.94.1 Detailed Description

template<typename T>
struct Catch::StringMaker< T * >

Definition at line 1292 of file catch.hpp.

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

- lib/catch.hpp

5.95 Catch::StringMaker< T[SZ]> Struct Template Reference

Static Public Member Functions

- static std::string **convert** (T const(&arr)[SZ])

5.95.1 Detailed Description

template<typename T, int SZ>
struct Catch::StringMaker< T[SZ]>

Definition at line 1517 of file catch.hpp.

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

- lib/catch.hpp

5.96 Catch::StringMaker< unsigned char > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (unsigned char c)

5.96.1 Detailed Description

template<>
struct Catch::StringMaker< unsigned char >

Definition at line 1273 of file catch.hpp.

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

- lib/catch.hpp

5.97 Catch::StringMaker< unsigned char[SZ]> Struct Template Reference

Static Public Member Functions

- static std::string **convert** (unsigned char const *str)

5.97.1 Detailed Description

template<int SZ>
struct Catch::StringMaker< unsigned char[SZ]>

Definition at line 1228 of file catch.hpp.

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

- lib/catch.hpp

5.98 Catch::StringMaker< unsigned int > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (unsigned int value)

5.98.1 Detailed Description

template<>
struct Catch::StringMaker< unsigned int >

Definition at line 1247 of file catch.hpp.

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

- lib/catch.hpp

5.99 Catch::StringMaker< unsigned long > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (unsigned long value)

5.99.1 Detailed Description

template<>
struct Catch::StringMaker< unsigned long >

Definition at line 1251 of file catch.hpp.

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

- lib/catch.hpp

5.100 Catch::StringMaker< unsigned long long > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (unsigned long long value)

5.100.1 Detailed Description

template<>
struct Catch::StringMaker< unsigned long long >

Definition at line 1255 of file catch.hpp.

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

- lib/catch.hpp

5.101 Catch::StringMaker< wchar_t * > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (wchar_t *str)

5.101.1 Detailed Description

template<>
struct Catch::StringMaker< wchar_t * >

Definition at line 1208 of file catch.hpp.

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

- lib/catch.hpp

5.102 Catch::StringMaker< wchar_t const * > Struct Template Reference

Static Public Member Functions

- static std::string **convert** (wchar_t const *str)

5.102.1 Detailed Description

template<>
struct Catch::StringMaker< wchar_t const * >

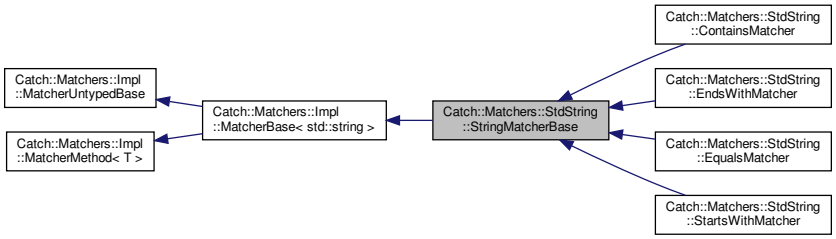
Definition at line 1204 of file catch.hpp.

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

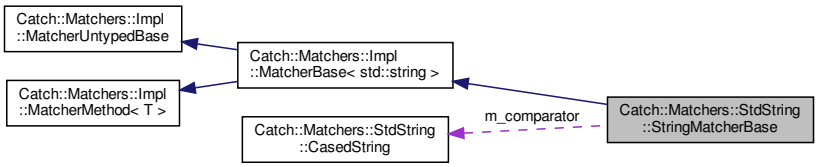
- lib/catch.hpp

5.103 Catch::Matchers::StdString::StringMatcherBase Struct Reference

Inheritance diagram for Catch::Matchers::StdString::StringMatcherBase:



Collaboration diagram for Catch::Matchers::StdString::StringMatcherBase:



Public Member Functions

- **StringMatcherBase** (std::string const &operation, CasedString const &comparator)
- std::string **describe** () const override

Public Attributes

- CasedString **m_comparator**
- std::string **m_operation**

Additional Inherited Members

5.103.1 Detailed Description

Definition at line 2877 of file catch.hpp.

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

- lib/catch.hpp

5.104 Catch::StringRef Class Reference

`#include <catch.hpp>`

Public Types

- using **size_type** = std::size_t

Public Member Functions

- **StringRef** (StringRef const &other) noexcept
- **StringRef** (StringRef &&other) noexcept
- **StringRef** (char const *rawChars) noexcept
- **StringRef** (char const *rawChars, size_type size) noexcept
- **StringRef** (std::string const &stdString) noexcept
- auto **operator=** (StringRef const &other) noexcept -> StringRef &
- **operator std::string** () const
- void **swap** (StringRef &other) noexcept
- auto **operator==** (StringRef const &other) const noexcept -> bool
- auto **operator!=** (StringRef const &other) const noexcept -> bool
- auto **operator[ ]** (size_type index) const noexcept -> char
- auto **empty** () const noexcept -> bool
- auto **size** () const noexcept -> size_type
- auto **numberOfCharacters** () const noexcept -> size_type
- auto **c_str** () const -> char const *
- auto **substr** (size_type start, size_type size) const noexcept -> StringRef
- auto **currentData** () const noexcept -> char const *

Friends

- struct **StringRefTestAccess**

5.104.1 Detailed Description

A non-owning string class (similar to the forthcoming std::string_view) Note that, because a StringRef may be a substring of another string, it may not be null terminated. c_str() must return a null terminated string, however, and so the StringRef will internally take ownership (taking a copy), if necessary. In theory this ownership is not externally visible - but it does mean (substring) StringRefs should not be shared between threads.

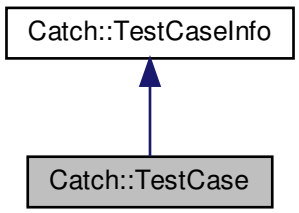
Definition at line 512 of file catch.hpp.

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

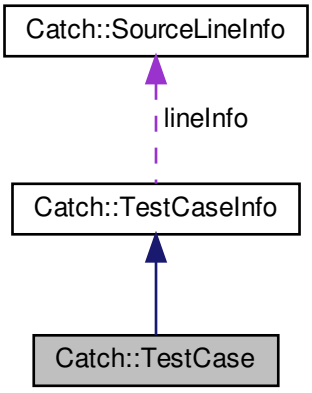
- lib/catch.hpp

5.105 Catch::TestCase Class Reference

Inheritance diagram for Catch::TestCase:



Collaboration diagram for Catch::TestCase:



Public Member Functions

- **TestCase** (ITestInvoker *testCase, TestCaseInfo &&info)
- TestCase **withName** (std::string const &_newName) const
- void **invoke** () const
- TestCaseInfo const & **getTestCaseInfo** () const
- bool **operator==** (TestCase const &other) const
- bool **operator<** (TestCase const &other) const

Additional Inherited Members

5.105.1 Detailed Description

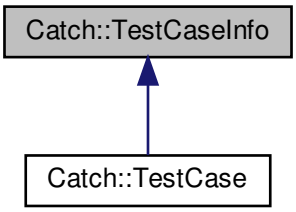
Definition at line 3519 of file catch.hpp.

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

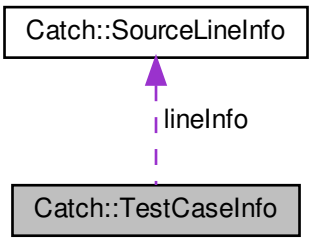
- lib/catch.hpp

5.106 Catch::TestCaseInfo Struct Reference

Inheritance diagram for Catch::TestCaseInfo:



Collaboration diagram for Catch::TestCaseInfo:



Public Types

- enum **SpecialProperties** {
  None = 0, **IsHidden** = 1 << 1, **ShouldFail** = 1 << 2, **MayFail** = 1 << 3,
  Throws = 1 << 4, **NonPortable** = 1 << 5, **Benchmark** = 1 << 6 }

Public Member Functions

- **TestCaseInfo** (std::string const &_name, std::string const &_className, std::string const &_description, std::vector< std::string > const &_tags, SourceLineInfo const &_lineInfo)
- bool **isHidden** () const
- bool **throws** () const
- bool **okToFail** () const
- bool **expectedToFail** () const
- std::string **tagsAsString** () const

Public Attributes

- std::string **name**
- std::string **className**
- std::string **description**
- std::vector< std::string > **tags**
- std::vector< std::string > **lcaseTags**
- SourceLineInfo **lineInfo**
- SpecialProperties **properties**

Friends

- void **setTags** (TestCaseInfo &testCaseInfo, std::vector< std::string > tags)

5.106.1 Detailed Description

Definition at line 3484 of file catch.hpp.

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

- lib/catch.hpp

5.107 Catch::TestFailureException Struct Reference

5.107.1 Detailed Description

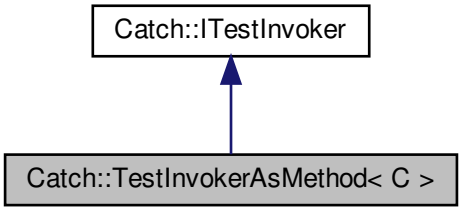
Definition at line 1884 of file catch.hpp.

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

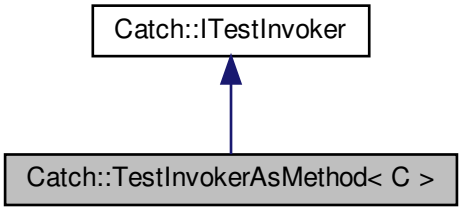
- lib/catch.hpp

5.108 Catch::TestInvokerAsMethod< C > Class Template Reference

Inheritance diagram for Catch::TestInvokerAsMethod< C >:



Collaboration diagram for Catch::TestInvokerAsMethod< C >:



Public Member Functions

- **TestInvokerAsMethod** (void(C::*testAsMethod)()) noexcept
- void **invoke** () const override

5.108.1 Detailed Description

template<typename C>
class Catch::TestInvokerAsMethod< C >

Definition at line 720 of file catch.hpp.

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

- lib/catch.hpp

5.109 Catch::Timer Class Reference

Public Member Functions

- void **start** ()
- auto **getElapsedNanoseconds** () const -> uint64_t
- auto **getElapsedMicroseconds** () const -> uint64_t
- auto **getElapsedMilliseconds** () const -> unsigned int
- auto **getElapsedSeconds** () const -> double

5.109.1 Detailed Description

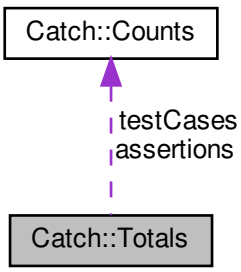
Definition at line 2249 of file catch.hpp.

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

- lib/catch.hpp

5.110 Catch::Totals Struct Reference

Collaboration diagram for Catch::Totals:



Public Member Functions

- Totals **operator-** (Totals const &other) const
- Totals & **operator+=** (Totals const &other)
- Totals **delta** (Totals const &prevTotals) const

Public Attributes

- int **error** = 0
- Counts **assertions**
- Counts **testCases**

5.110.1 Detailed Description

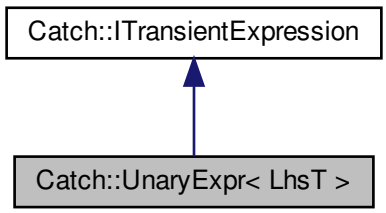
Definition at line 2197 of file catch.hpp.

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

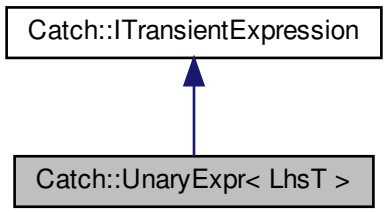
- lib/catch.hpp

5.111 Catch::UnaryExpr< LhsT > Class Template Reference

Inheritance diagram for Catch::UnaryExpr< LhsT >:



Collaboration diagram for Catch::UnaryExpr< LhsT >:



Public Member Functions

- **UnaryExpr** (LhsT lhs)

Additional Inherited Members

5.111.1 Detailed Description

template<typename LhsT>
class Catch::UnaryExpr< LhsT >

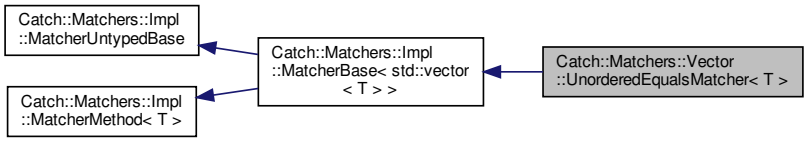
Definition at line 1701 of file catch.hpp.

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

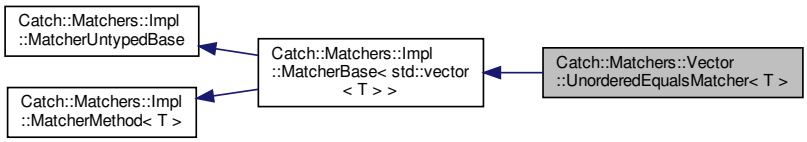
- lib/catch.hpp

5.112 Catch::Matchers::Vector::UnorderedEqualsMatcher< T > Struct Template Reference

Inheritance diagram for Catch::Matchers::Vector::UnorderedEqualsMatcher< T >:



Collaboration diagram for Catch::Matchers::Vector::UnorderedEqualsMatcher< T >:



Public Member Functions

- **UnorderedEqualsMatcher** (std::vector< T > const &target)
- bool **match** (std::vector< T > const &vec) const override
- std::string **describe** () const override

Additional Inherited Members

5.112.1 Detailed Description

template<typename T>
struct Catch::Matchers::Vector::UnorderedEqualsMatcher< T >

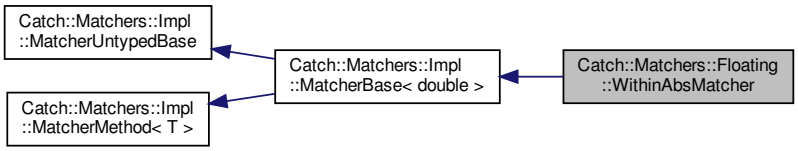
Definition at line 3032 of file catch.hpp.

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

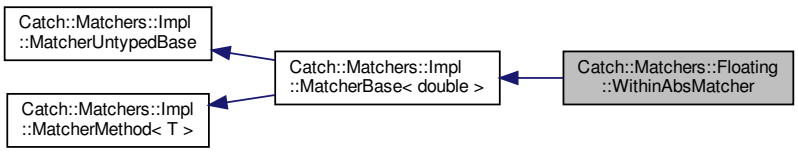
- lib/catch.hpp

5.113 Catch::Matchers::Floating::WithinAbsMatcher Struct Reference

Inheritance diagram for Catch::Matchers::Floating::WithinAbsMatcher:



Collaboration diagram for Catch::Matchers::Floating::WithinAbsMatcher:



Public Member Functions

- **WithinAbsMatcher** (double target, double margin)
- bool **match** (double const &matchee) const override
- std::string **describe** () const override

Additional Inherited Members

5.113.1 Detailed Description

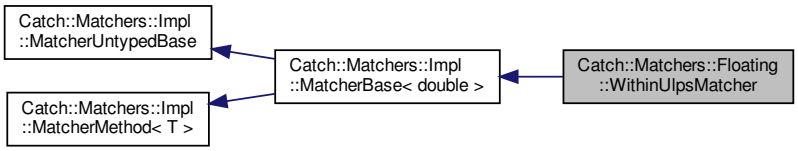
Definition at line 2779 of file catch.hpp.

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

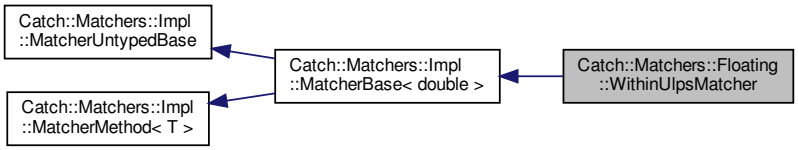
- lib/catch.hpp

5.114 Catch::Matchers::Floating::WithinUlpsMatcher Struct Reference

Inheritance diagram for Catch::Matchers::Floating::WithinUlpsMatcher:



Collaboration diagram for Catch::Matchers::Floating::WithinUlpsMatcher:



Public Member Functions

- **WithinUlpsMatcher** (double target, int ulps, FloatingPointKind baseType)
- bool **match** (double const &matchee) const override
- std::string **describe** () const override

Additional Inherited Members

5.114.1 Detailed Description

Definition at line 2788 of file catch.hpp.

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

- lib/catch.hpp

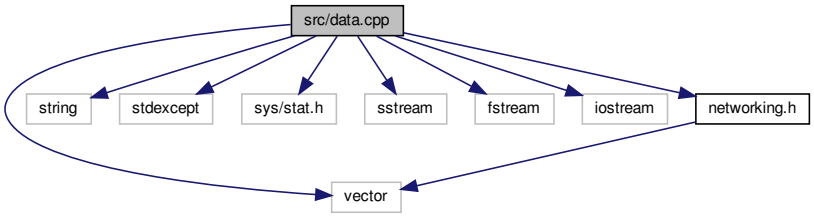
Chapter 6

File Documentation

6.1 src/data.cpp File Reference

```
#include <vector>
#include <string>
#include <stdexcept>
#include <sys/stat.h>
#include <sstream>
#include <fstream>
#include <iostream>
#include "networking.h"
```

Include dependency graph for data.cpp:



Macros

- #define **DEFAULT_FILENAME** "matrix.in"

Functions

- bool fileExists (const std::string &name)

  Check if a file with the given name exists.
- std::string inputFileName (int argc, char **argv)

  Decide which filename should be used to read the data from.
- void ReadData (std::vector< std::vector< int >> *data, int *arraySize, int argc, char **argv)

Determines the source to read from, reads and writes the data to a 2D vector.

- int ** vector2DToArray2D (std::vector< std::vector< int >> v2d)

  Returns a 2D Array with the data of the passed 2D Vector.
- int * array2DTo1DRowMajor (int **arr2d, int m, int n)

  Converts a 2D array into an 1D array in row-major order.
- std::vector< std::vector< int > > arrayRowMajorTo2DVector (int *arr, int m, int n)

  Converts an array in row-major order to a 2D vector.
- void prepareData (std::vector< std::vector< int >> data, int *&dataArray)

  Convert data in 2D vector to an 1D array with the data of the original vector in row-major order for use with MPI communication methods.
- int checkCriteriaLocal (std::vector< std::vector< int >> localData, int arraySize, int myRank, int processes, int *maxLocal)

  Check if the lines present in a process meet the strictly diagonally dominant criteria.
- int calculateBCell (int a, int max, int x, int y)

  Calculate the value of a cell of the B matrix.
- std::vector< std::vector< int > > calculateBLocal (std::vector< std::vector< int >> localData, int max, int arraySize, int myRank, int processes, int *min, int *minX, int *minY)

  Calculate the lines of B assigned to a process.
- void printB (int *b, int arraySize)

  Prints matrix B.

6.1.1 Detailed Description

Author

Konstantinos Kamaropoulos (kamaropoulos@outlook.com)

Version

0.1

Date

2018-12-27

Copyright

Definition in file data.cpp.

6.1.2 Function Documentation

6.1.2.1 array2DTo1DRowMajor()

```
int* array2DTo1DRowMajor (
            int ** arr2d,
            int m,
            int n )
```

Converts a 2D array into an 1D array in row-major order.

Parameters

arr2d	[in] A 2D array
m	[in] The number of lines in the 2D array
n	[in] The number of columns in the 2D array

Returns

int* A pointer to an array with the data of the passed 2D array in row-major order

Definition at line 165 of file data.cpp.

```
00166 {
00167     // Create a pointer for an array
00168     int *arr = new int[m * n];
00169
00170     for (int i = 0; i < m; i++)
00171     {
00172         for (int j = 0; j < n; j++)
00173         {
00174             arr[i * n + j] = arr2d[i][j];
00175         }
00176     }
00177
00178     return arr;
00179 }
```

6.1.2.2 arrayRowMajorTo2DVector()

```
std::vector<std::vector<int> > arrayRowMajorTo2DVector (
            int * arr,
            int m,
            int n )
```

Converts an array in row-major order to a 2D vector.

Parameters

arr	[in] A pointer to an array
m	[in] The number of lines in the array
n	[in] The number of columns in the array

Returns

std::vector<std::vector<int>> A vector with the row-major order data from the passed array

Definition at line 190 of file data.cpp.

```
00191 {
00192     std::vector<std::vector<int» vec;
00193
00194     int index = 0;
00195     for (int i = 0; i < m; i++)
00196     {
00197         std::vector<int> vecLine;
00198
00199         for (int j = 0; j < n; j++)
00200         {
00201             vecLine.push_back(arr[index++]);
00202         }
```

```
00203
00204         vec.push_back(vecLine);
00205     }
00206
00207     return vec;
00208 }
```

6.1.2.3 calculateBCell()

```
int calculateBCell (
              int a,
              int max,
              int x,
              int y )
```

Calculate the value of a cell of the B matrix.

Parameters

a	[in] The element of matrix A in the same position for which we want to calculate the value of B
max	[in] The max element of the A matrix
x	[in] The line we want to find an element for
y	[in] The column we want to find an element for

Returns

int The calculated value of the requested cell

Definition at line 300 of file data.cpp.

```
00301 {
00302     if (x == y)
00303         return max;
00304     return max - abs(a);
00305 }
```

Here is the caller graph for this function:

6.1.2.4 calculateBLocal()

```
std::vector<std::vector<int> > calculateBLocal (
            std::vector< std::vector< int >> localData,
            int max,
            int arraySize,
            int myRank,
            int processes,
            int * min,
            int * minX,
            int * minY )
```

Calculate the lines of B assigned to a process.

Parameters

localData	[in] The local lines of matrix A
max	[in] The max value found on matrix A
arraySize	[in] The total number of lines in all processes
myRank	[in] The rank of the process
processes	[in] The total number of processes
min	[out] The min item found on all lines calculated by the current process
minX	[out] The X coordinate of the min item on all lines calculated by the current process
minY	[out] The Y coordinate of the min item on all lines calculated by the current process

Returns

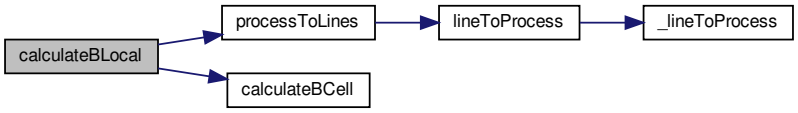
std::vector<std::vector<int>> A 2D vector holding the lines calculated by the current process

Definition at line 320 of file data.cpp.

```
00321 {
00322     std::vector<std::vector<int» localB;
00323     std::vector<int> lines = processToLines(myRank, arraySize, processes);
00324
00325     *minX = -1;
00326     *minY = -1;
00327
00328     for (int i = 0; i < lines.size(); i++)
00329     {
00330         std::vector<int> line;
00331
00332         for (int j = 0; j < arraySize; j++)
00333         {
00334
00335             line.push_back(calculateBCell(localData[i][j], max, lines[i], j));
00336
00337             if ((i == 0) && (j == 0))
00338             {
00339                 *min = line[0];
00340                 *minX = 0;
00341                 *minY = 0;
00342             }
00343             else
00344             {
00345                 if (line[j] < *min)
00346                 {
00347                     *min = line[j];
00348                     *minX = lines[i];
00349                     *minY = j;
00350                 }
00351             }
00352         }
00353         localB.push_back(line);
00354     }
00355
00356     return localB;
```

```
00357 }
```

Here is the call graph for this function:



6.1.2.5 checkCriteriaLocal()

```
int checkCriteriaLocal (
            std::vector< std::vector< int >> localData,
            int arraySize,
            int myRank,
            int processes,
            int * maxLocal )
```

Check if the lines present in a process meet the strictly diagonally dominant criteria.

Parameters

localData	[in] A 2D vector holding the lines the process has
arraySize	[in] The total number of lines in all processes
myRank	[in] The rank of the process
processes	[in] The total number of processes
maxLocal	[out] The local max found in the local lines

Returns

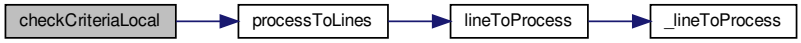
int 0 if the local lines meet the criteria; 1 if the local lines do not meet the criteria

Definition at line 235 of file data.cpp.

```
00236 {
00237
00238     int result = 0;
00239
00240     int maxLocalTemp;
00241
00242     std::vector<int> lines = processToLines(myRank, arraySize, processes);
00243
00244     // For each line:
00245     for (int i = 0; i < localData.size(); i++)
00246     {
00247
00248         int line = lines[i];
00249         int diagonalElement = localData[i][line];
00250
00251         if (i == 0)
00252         {
00253             maxLocalTemp = diagonalElement;
00254         }
```

```
00255         else
00256         {
00257             maxLocalTemp = (abs(diagonalElement) > maxLocalTemp) ? diagonalElement : maxLocalTemp;
00258         }
00259
00260         // Calculate sum
00261         int sum = 0;
00262         for (int j = 0; j < arraySize; j++)
00263         {
00264             if (j != line)
00265             {
00266                 sum += abs(localData[i][j]);
00267             }
00268         }
00269
00270         bool ok = diagonalElement >= sum;
00271
00272         if (ok)
00273         {
00274             result += 1;
00275         }
00276     }
00277
00278     *maxLocal = maxLocalTemp;
00279
00280     if (result == lines.size())
00281     {
00282         return 1;
00283     }
00284     else
00285     {
00286         return 0;
00287     }
00288 }
```

Here is the call graph for this function:



6.1.2.6 fileExists()

```
bool fileExists (
            const std::string & name )  [inline]
```

Check if a file with the given name exists.

Parameters

name	[in] The name of the file to check

Returns

true The file exists
false The file doesn't exist

Definition at line 35 of file data.cpp.

```
00036 {
```

```
00037     struct stat buffer;
00038     return (stat(name.c_str(), &buffer) == 0);
00039 }
```

6.1.2.7 inputFileName()

```
std::string inputFileName (
              int argc,
              char ** argv )
```

Decide which filename should be used to read the data from.

Parameters

argc	[in] The number of command line arguments
argv	[in] The command line arguments

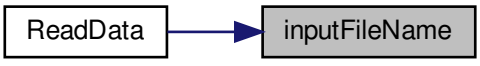
Returns

std::string The filename to use for reading the data

Definition at line 48 of file data.cpp.

```
00049 {
00050     std::string filename;
00051     // Check if file name was passed
00052     switch (argc)
00053     {
00054     case 1:
00055         // No parameter passed, fallback to default filename
00056         return DEFAULT_FILENAME;
00057         break;
00058
00059     case 2:
00060         // We got one parameter, check if it's an existing file
00061         filename = argv[1];
00062         if (fileExists(filename))
00063         {
00064             return filename;
00065         }
00066         else
00067         {
00068             throw std::runtime_error("Could not open file \"" + filename + "\"");
00069         }
00070
00071     default:
00072         throw std::runtime_error("Incorrect command usage!");
00073         break;
00074     }
00075 }
```

Here is the caller graph for this function:



6.1.2.8 prepareData()

```
void prepareData (
            std::vector< std::vector< int >> data,
            int *& dataArray )
```

Convert data in 2D vector to an 1D array with the data of the original vector in row-major order for use with MPI communication methods.

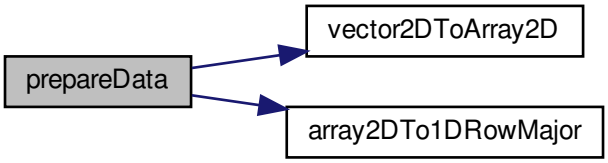
Parameters

data	[in] A 2D vector
dataArray	[in] A pointer to an array with the size of the data already allocated

Definition at line 217 of file data.cpp.

```
00218 {
00219     int **arr2D = vector2DToArray2D(data);
00220     dataArray = array2DTo1DRowMajor(arr2D, data.size(), data[0].size());
00221 }
```

Here is the call graph for this function:



6.1.2.9 printB()

```
void printB (
            int * b,
            int arraySize )
```

Prints matrix B.

Parameters

b	[in] A pointer to an array in row-major order
arraySize	[in] The total number of lines in all processes

Definition at line 365 of file data.cpp.

```
00366 {
00367     for (int i = 0; i < arraySize; i++)
```

```
00368     {
00369         for (int j = 0; j < arraySize; j++)
00370         {
00371 
00372             std::cout « b[(i * arraySize) + j] « " ";
00373         }
00374         std::cout « std::endl;
00375     }
00376 }
```

6.1.2.10 ReadData()

```
void ReadData (
            std::vector< std::vector< int >> * data,
            int * arraySize,
            int argc,
            char ** argv )
```

Determines the source to read from, reads and writes the data to a 2D vector.

Parameters

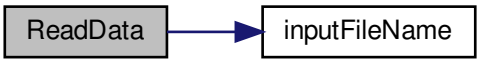
data	[out] A pointer to an <std::vector<std::vector<int>> collection to write the data to
argc	[in] The number of command line arguments
argv	[in] The command line arguments

Definition at line 84 of file data.cpp.

```
00085 {
00086     std::string filename;
00087 
00088     try
00089     {
00090         filename = inputFileName(argc, argv);
00091     }
00092     catch (std::exception &e)
00093     {
00094         std::cout « "\033[1;31m"
00095                   « "Error: " « e.what() « "\033[0m"
00096                   « std::endl;
00097     }
00098 
00099     // An input file stream used to read data from config.txt.
00100     std::ifstream inputFile(filename);
00101 
00102     // If the file is opened sucessfully
00103     if (inputFile.is_open())
00104     {
00105         // A temporary variable to store numbers from the input file.
00106         int tmpNumber;
00107 
00108         std::string line;
00109 
00110         // While inputFile still has lines, put them on line.
00111         while (std::getline(inputFile, line))
00112         {
00113             std::istringstream iss(line);
00114             std::vector<int> tempv;
00115 
00116             while (iss » tmpNumber)
00117             {
00118                 tempv.push_back(tmpNumber);
00119             }
00120 
00121             data->push_back(tempv);
00122         }
00123 
00124         // We are at the end of the file, close it as we won't need it anymore.
00125         inputFile.close();
00126     }
00127 
```

```
00128     *arraySize = data->size();
00129 }
```

Here is the call graph for this function:



6.1.2.11 vector2DToArray2D()

```
int** vector2DToArray2D (
            std::vector< std::vector< int >> v2d )
```

Returns a 2D Array with the data of the passed 2D Vector.

Parameters

v2d	[in] The 2D Vector to get the data from

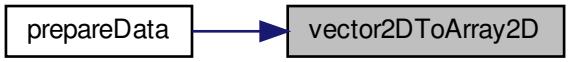
Returns

int** A pointer to the 2D array with the data from the vector

Definition at line 137 of file data.cpp.

```
00138 {
00139     // Create a pointer for a 2D array
00140     int **arr2d = new int *[v2d.size()];
00141 
00142     // For each line in the vector:
00143     for (int i = 0; i < v2d.size(); i++)
00144     {
00145         arr2d[i] = new int[v2d[i].size()];
00146 
00147         for (int j = 0; j < v2d[i].size(); j++)
00148         {
00149             arr2d[i][j] = v2d[i][j];
00150         }
00151     }
00152 
00153     return arr2d;
00154 }
```

Here is the caller graph for this function:



6.2 data.cpp

```
00001
00012 #define DEFAULT_FILENAME "matrix.in"
00013
00014 #include <vector>
00015 #include <string>
00016
00017 #include <stdexcept>
00018
00019 #include <sys/stat.h>
00020
00021 #include <sstream>
00022 #include <fstream>
00023
00024 #include <iostream>
00025
00026 #include "networking.h"
00027
00035 inline bool fileExists(const std::string &name)
00036 {
00037     struct stat buffer;
00038     return (stat(name.c_str(), &buffer) == 0);
00039 }
00040
00048 std::string inputFileName(int argc, char **argv)
00049 {
00050     std::string filename;
00051     // Check if file name was passed
00052     switch (argc)
00053     {
00054     case 1:
00055         // No parameter passed, fallback to default filename
00056         return DEFAULT_FILENAME;
00057         break;
00058
00059     case 2:
00060         // We got one parameter, check if it's an existing file
00061         filename = argv[1];
00062         if (fileExists(filename))
00063         {
00064             return filename;
00065         }
00066         else
00067         {
00068             throw std::runtime_error("Could not open file \"" + filename + "\"");
00069         }
00070
00071     default:
00072         throw std::runtime_error("Incorrect command usage!");
00073         break;
00074     }
00075 }
00076
00084 void ReadData(std::vector<std::vector<int» *data, int *arraySize, int argc, char **argv)
00085 {
00086     std::string filename;
00087
00088     try
00089     {
00090         filename = inputFileName(argc, argv);
00091     }
00092     catch (std::exception &e)
00093     {
00094         std::cout « "\033[1;31m"
00095                   « "Error: " « e.what() « "\033[0m"
```

```
00096                   « std::endl;
00097     }
00098
00099     // An input file stream used to read data from config.txt.
00100     std::ifstream inputFile(filename);
00101
00102     // If the file is opened sucessfully
00103     if (inputFile.is_open())
00104     {
00105         // A temporary variable to store numbers from the input file.
00106         int tmpNumber;
00107
00108         std::string line;
00109
00110         // While inputFile still has lines, put them on line.
00111         while (std::getline(inputFile, line))
00112         {
00113             std::istringstream iss(line);
00114             std::vector<int> tempv;
00115
00116             while (iss » tmpNumber)
00117             {
00118                 tempv.push_back(tmpNumber);
00119             }
00120
00121             data->push_back(tempv);
00122         }
00123
00124         // We are at the end of the file, close it as we won't need it anymore.
00125         inputFile.close();
00126     }
00127
00128     *arraySize = data->size();
00129 }
00130
00137 int **vector2DToArray2D(std::vector<std::vector<int» v2d)
00138 {
00139     // Create a pointer for a 2D array
00140     int **arr2d = new int *[v2d.size()];
00141
00142     // For each line in the vector:
00143     for (int i = 0; i < v2d.size(); i++)
00144     {
00145         arr2d[i] = new int[v2d[i].size()];
00146
00147         for (int j = 0; j < v2d[i].size(); j++)
00148         {
00149             arr2d[i][j] = v2d[i][j];
00150         }
00151     }
00152
00153     return arr2d;
00154 }
00155
00165 int *array2DTo1DRowMajor(int **arr2d, int m, int n)
00166 {
00167     // Create a pointer for an array
00168     int *arr = new int[m * n];
00169
00170     for (int i = 0; i < m; i++)
00171     {
00172         for (int j = 0; j < n; j++)
00173         {
00174             arr[i * n + j] = arr2d[i][j];
00175         }
00176     }
00177
00178     return arr;
00179 }
00180
00190 std::vector<std::vector<int» arrayRowMajorTo2DVector(int *arr, int m, int n)
00191 {
00192     std::vector<std::vector<int» vec;
00193
00194     int index = 0;
00195     for (int i = 0; i < m; i++)
00196     {
00197         std::vector<int> vecLine;
00198
00199         for (int j = 0; j < n; j++)
00200         {
00201             vecLine.push_back(arr[index++]);
00202         }
00203
00204         vec.push_back(vecLine);
00205     }
00206
```

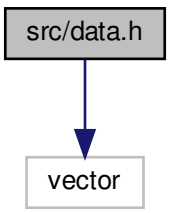
```
00207     return vec;
00208 }
00209
00217 void prepareData(std::vector<std::vector<int>> data, int *&dataArray)
00218 {
00219     int **arr2D = vector2DToArray2D(data);
00220     dataArray = array2DTo1DRowMajor(arr2D, data.size(), data[0].size());
00221 }
00222
00235 int checkCriteriaLocal(std::vector<std::vector<int>> localData, int arraySize, int myRank, int
      processes, int *maxLocal)
00236 {
00237
00238     int result = 0;
00239
00240     int maxLocalTemp;
00241
00242     std::vector<int> lines = processToLines(myRank, arraySize, processes);
00243
00244     // For each line:
00245     for (int i = 0; i < localData.size(); i++)
00246     {
00247
00248         int line = lines[i];
00249         int diagonalElement = localData[i][line];
00250
00251         if (i == 0)
00252         {
00253             maxLocalTemp = diagonalElement;
00254         }
00255         else
00256         {
00257             maxLocalTemp = (abs(diagonalElement) > maxLocalTemp) ? diagonalElement : maxLocalTemp;
00258         }
00259
00260         // Calculate sum
00261         int sum = 0;
00262         for (int j = 0; j < arraySize; j++)
00263         {
00264             if (j != line)
00265             {
00266                 sum += abs(localData[i][j]);
00267             }
00268         }
00269
00270         bool ok = diagonalElement >= sum;
00271
00272         if (ok)
00273         {
00274             result += 1;
00275         }
00276     }
00277
00278     *maxLocal = maxLocalTemp;
00279
00280     if (result == lines.size())
00281     {
00282         return 1;
00283     }
00284     else
00285     {
00286         return 0;
00287     }
00288 }
00289
00300 int calculateBCell(int a, int max, int x, int y)
00301 {
00302     if (x == y)
00303         return max;
00304     return max - abs(a);
00305 }
00306
00320 std::vector<std::vector<int>> calculateBLocal(std::vector<std::vector<int>> localData, int max, int
      arraySize, int myRank, int processes, int *min, int *minX, int *minY)
00321 {
00322     std::vector<std::vector<int>> localB;
00323     std::vector<int> lines = processToLines(myRank, arraySize, processes);
00324
00325     *minX = -1;
00326     *minY = -1;
00327
00328     for (int i = 0; i < lines.size(); i++)
00329     {
00330         std::vector<int> line;
00331
00332         for (int j = 0; j < arraySize; j++)
00333         {
```

```
00334
00335             line.push_back(calculateBCell(localData[i][j], max, lines[i], j));
00336
00337             if ((i == 0) && (j == 0))
00338             {
00339                 *min = line[0];
00340                 *minX = 0;
00341                 *minY = 0;
00342             }
00343             else
00344             {
00345                 if (line[j] < *min)
00346                 {
00347                     *min = line[j];
00348                     *minX = lines[i];
00349                     *minY = j;
00350                 }
00351             }
00352         }
00353         localB.push_back(line);
00354     }
00355
00356     return localB;
00357 }
00358
00365 void printB(int *b, int arraySize)
00366 {
00367     for (int i = 0; i < arraySize; i++)
00368     {
00369         for (int j = 0; j < arraySize; j++)
00370         {
00371
00372             std::cout « b[(i * arraySize) + j] « " ";
00373         }
00374         std::cout « std::endl;
00375     }
00376 }
```

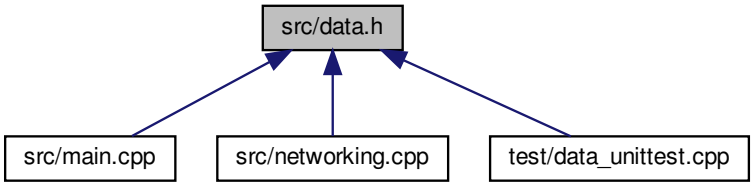
6.3 src/data.h File Reference

`#include <vector>`

Include dependency graph for data.h:



This graph shows which files directly or indirectly include this file:



Functions

- void ReadData (std::vector< std::vector< int >> *, int *, int, char **)

  Determines the source to read from, reads and writes the data to a 2D vector.
- int ** **vector2DToArray2D** (std::vector< std::vector< int >>)
- int * array2DTo1DRowMajor (int **arr2d, int m, int n)

  Converts a 2D array into an 1D array in row-major order.
- std::vector< std::vector< int > > arrayRowMajorTo2DVector (int *arr, int m, int n)

  Converts an array in row-major order to a 2D vector.
- void prepareData (std::vector< std::vector< int >> data, int *&dataArray)

  Convert data in 2D vector to an 1D array with the data of the original vector in row-major order for use with MPI communication methods.
- int checkCriteriaLocal (std::vector< std::vector< int >> localData, int arraySize, int myRank, int processes, int *maxLocal)

  Check if the lines present in a process meet the strictly diagonally dominant criteria.
- std::vector< std::vector< int > > calculateBLocal (std::vector< std::vector< int >> localData, int max, int arraySize, int myRank, int processes, int *min, int *minX, int *minY)

  Calculate the lines of B assigned to a process.
- void printB (int *b, int arraySize)

  Prints matrix B.

6.3.1 Detailed Description

Author

Konstantinos Kamaropoulos (kamaropoulos@outlook.com)

Version

0.1

Date

2018-12-27

Copyright

Copyright (c) 2018

Definition in file data.h.

6.3.2 Function Documentation

6.3.2.1 array2DTo1DRowMajor()

```
int* array2DTo1DRowMajor (
            int ** arr2d,
            int m,
            int n )
```

Converts a 2D array into an 1D array in row-major order.

Parameters

arr2d	[in] A 2D array
m	[in] The number of lines in the 2D array
n	[in] The number of columns in the 2D array

Returns

int* A pointer to an array with the data of the passed 2D array in row-major order

Definition at line 165 of file data.cpp.

```
00166 {
00167     // Create a pointer for an array
00168     int *arr = new int[m * n];
00169
00170     for (int i = 0; i < m; i++)
00171     {
00172         for (int j = 0; j < n; j++)
00173         {
00174             arr[i * n + j] = arr2d[i][j];
00175         }
00176     }
00177
00178     return arr;
00179 }
```

6.3.2.2 arrayRowMajorTo2DVector()

```
std::vector<std::vector<int> > arrayRowMajorTo2DVector (
            int * arr,
            int m,
            int n )
```

Converts an array in row-major order to a 2D vector.

Parameters

arr	[in] A pointer to an array
m	[in] The number of lines in the array
n	[in] The number of columns in the array

Returns

std::vector<std::vector<int>> A vector with the row-major order data from the passed array

Definition at line 190 of file data.cpp.

```
00191 {
00192     std::vector<std::vector<int» vec;
00193
00194     int index = 0;
00195     for (int i = 0; i < m; i++)
00196     {
00197         std::vector<int> vecLine;
00198
00199         for (int j = 0; j < n; j++)
00200         {
00201             vecLine.push_back(arr[index++]);
00202         }
00203
00204         vec.push_back(vecLine);
00205     }
00206
00207     return vec;
00208 }
```

6.3.2.3 calculateBLocal()

```
std::vector<std::vector<int> > calculateBLocal (
             std::vector< std::vector< int >> localData,
             int max,
             int arraySize,
             int myRank,
             int processes,
             int * min,
             int * minX,
             int * minY )
```

Calculate the lines of B assigned to a process.

Parameters

localData	[in] The local lines of matrix A
max	[in] The max value found on matrix A
arraySize	[in] The total number of lines in all processes
myRank	[in] The rank of the process
processes	[in] The total number of processes
min	[out] The min item found on all lines calculated by the current process
minX	[out] The X coordinate of the min item on all lines calculated by the current process
minY	[out] The Y coordinate of the min item on all lines calculated by the current process

Returns

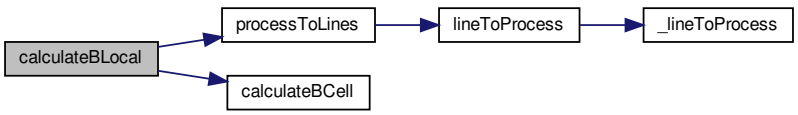
std::vector<std::vector<int>> A 2D vector holding the lines calculated by the current process

Definition at line 320 of file data.cpp.

```
00321 {
00322     std::vector<std::vector<int» localB;
00323     std::vector<int> lines = processToLines(myRank, arraySize, processes);
00324
```

```
00325     *minX = -1;
00326     *minY = -1;
00327
00328     for (int i = 0; i < lines.size(); i++)
00329     {
00330         std::vector<int> line;
00331
00332         for (int j = 0; j < arraySize; j++)
00333         {
00334
00335             line.push_back(calculateBCell(localData[i][j], max, lines[i], j));
00336
00337             if ((i == 0) && (j == 0))
00338             {
00339                 *min = line[0];
00340                 *minX = 0;
00341                 *minY = 0;
00342             }
00343             else
00344             {
00345                 if (line[j] < *min)
00346                 {
00347                     *min = line[j];
00348                     *minX = lines[i];
00349                     *minY = j;
00350                 }
00351             }
00352         }
00353         localB.push_back(line);
00354     }
00355
00356     return localB;
00357 }
```

Here is the call graph for this function:



6.3.2.4 checkCriteriaLocal()

```
int checkCriteriaLocal (
             std::vector< std::vector< int >> localData,
             int arraySize,
             int myRank,
             int processes,
             int * maxLocal )
```

Check if the lines present in a process meet the strictly diagonally dominant criteria.

Parameters

localData	[in] A 2D vector holding the lines the process has
arraySize	[in] The total number of lines in all processes
myRank	[in] The rank of the process
processes	[in] The total number of processes
maxLocal	[out] The local max found in the local lines

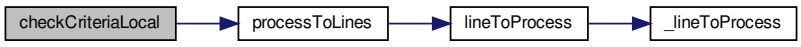
Returns

int 0 if the local lines meet the criteria; 1 if the local lines do not meet the criteria

Definition at line 235 of file data.cpp.

```
00236 {
00237
00238     int result = 0;
00239
00240     int maxLocalTemp;
00241
00242     std::vector<int> lines = processToLines(myRank, arraySize, processes);
00243
00244     // For each line:
00245     for (int i = 0; i < localData.size(); i++)
00246     {
00247
00248         int line = lines[i];
00249         int diagonalElement = localData[i][line];
00250
00251         if (i == 0)
00252         {
00253             maxLocalTemp = diagonalElement;
00254         }
00255         else
00256         {
00257             maxLocalTemp = (abs(diagonalElement) > maxLocalTemp) ? diagonalElement : maxLocalTemp;
00258         }
00259
00260         // Calculate sum
00261         int sum = 0;
00262         for (int j = 0; j < arraySize; j++)
00263         {
00264             if (j != line)
00265             {
00266                 sum += abs(localData[i][j]);
00267             }
00268         }
00269
00270         bool ok = diagonalElement >= sum;
00271
00272         if (ok)
00273         {
00274             result += 1;
00275         }
00276     }
00277
00278     *maxLocal = maxLocalTemp;
00279
00280     if (result == lines.size())
00281     {
00282         return 1;
00283     }
00284     else
00285     {
00286         return 0;
00287     }
00288 }
```

Here is the call graph for this function:



6.3.2.5 prepareData()

```
void prepareData (
            std::vector< std::vector< int >> data,
            int *& dataArray )
```

Convert data in 2D vector to an 1D array with the data of the original vector in row-major order for use with MPI communication methods.

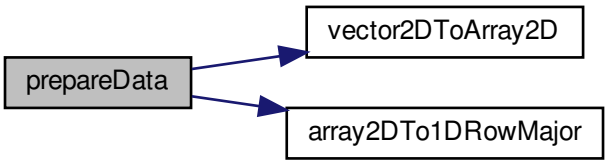
Parameters

data	[in] A 2D vector
dataArray	[in] A pointer to an array with the size of the data already allocated

Definition at line 217 of file data.cpp.

```
00218 {
00219     int **arr2D = vector2DToArray2D(data);
00220     dataArray = array2DTo1DRowMajor(arr2D, data.size(), data[0].size());
00221 }
```

Here is the call graph for this function:



6.3.2.6 printB()

```
void printB (
            int * b,
            int arraySize )
```

Prints matrix B.

Parameters

b	[in] A pointer to an array in row-major order
arraySize	[in] The total number of lines in all processes

Definition at line 365 of file data.cpp.

```
00366 {
00367     for (int i = 0; i < arraySize; i++)
00368     {
00369         for (int j = 0; j < arraySize; j++)
00370         {
00371 
00372             std::cout « b[(i * arraySize) + j] « " ";
00373         }
00374         std::cout « std::endl;
00375     }
00376 }
```

6.3.2.7 ReadData()

```
void ReadData (
            std::vector< std::vector< int >> * data,
            int * arraySize,
            int argc,
            char ** argv )
```

Determines the source to read from, reads and writes the data to a 2D vector.

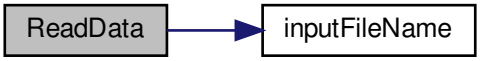
Parameters

data	[out] A pointer to an <std::vector<std::vector<int>> collection to write the data to
argc	[in] The number of command line arguments
argv	[in] The command line arguments

Definition at line 84 of file data.cpp.

```
00085 {
00086     std::string filename;
00087
00088     try
00089     {
00090         filename = inputFileName(argc, argv);
00091     }
00092     catch (std::exception &e)
00093     {
00094         std::cout « "\033[1;31m"
00095                   « "Error: " « e.what() « "\033[0m"
00096                   « std::endl;
00097     }
00098
00099     // An input file stream used to read data from config.txt.
00100     std::ifstream inputFile(filename);
00101
00102     // If the file is opened sucessfully
00103     if (inputFile.is_open())
00104     {
00105         // A temporary variable to store numbers from the input file.
00106         int tmpNumber;
00107
00108         std::string line;
00109
00110         // While inputFile still has lines, put them on line.
00111         while (std::getline(inputFile, line))
00112         {
00113             std::istringstream iss(line);
00114             std::vector<int> tempv;
00115
00116             while (iss » tmpNumber)
00117             {
00118                 tempv.push_back(tmpNumber);
00119             }
00120
00121             data->push_back(tempv);
00122         }
00123
00124         // We are at the end of the file, close it as we won't need it anymore.
00125         inputFile.close();
00126     }
00127
00128     *arraySize = data->size();
00129 }
```

Here is the call graph for this function:



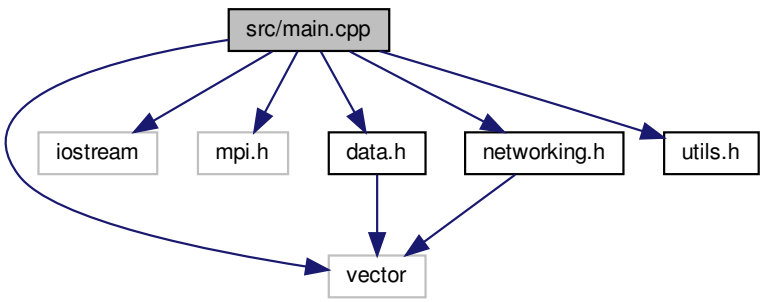
6.4 data.h

```
00001
00012 #include <vector>
00013
00014 void ReadData(std::vector<std::vector<int>> *, int*, int, char **);
00015
00016 int **vector2DToArray2D(std::vector<std::vector<int>>);
00017
00018 int *array2DTo1DRowMajor(int **arr2d, int m, int n);
00019
00020 std::vector<std::vector<int>> arrayRowMajorTo2DVector(int *arr, int m, int n);
00021
00022 void prepareData(std::vector<std::vector<int>> data, int *&dataArray);
00023
00024 int checkCriteriaLocal(std::vector<std::vector<int>> localData, int arraySize, int myRank, int
      processes, int *maxLocal);
00025
00026 std::vector<std::vector<int>> calculateBLocal(std::vector<std::vector<int>> localData, int max, int
      arraySize, int myRank, int processes, int *min, int *minX, int *minY);
00027
00028 void printB(int *b, int arraySize);
```

6.5 src/main.cpp File Reference

```
#include <vector>
#include <iostream>
#include "mpi.h"
#include "data.h"
#include "networking.h"
#include "utils.h"
```

Include dependency graph for main.cpp:



Macros

- #define **ROOT_PROCESS** 0
- #define **ifRoot**(callback) if (myRank == ROOT_PROCESS) callback

Functions

- int **main** (int argc, char ∗∗argv)

6.5.1 Detailed Description

Author

Konstantinos Kamaropoulos (kamaropoulos@outlook.com)

Version

0.1

Date

2018-12-25

Copyright

Definition in file main.cpp.

6.6 main.cpp

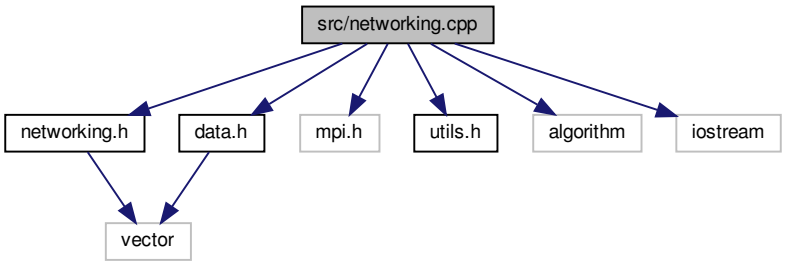
```
00001
00012 #include <vector>
00013
00014 #include <iostream>
00015
00016 #include "mpi.h"
00017
00018 #include "data.h"
00019 #include "networking.h"
00020
00021 #define ROOT_PROCESS 0
00022
00023 #include "utils.h"
00024
00025 int main(int argc, char **argv)
00026 {
00027     // The number of processes the code is running on.
00028     int processes;
00029     // The rank of the current process.
00030     int myRank;
00031     std::vector<std::vector<int» data;
00032     std::vector<std::vector<int» localData;
00033     // The size of the original array
00034     int arraySize = 0;
00035     int max;
00036
00037     int *displs, *scounts;
00038
00039     // Initialize MPI.
00040     MPI_Init(&argc, &argv);
```

```
00041
00042     // Get running processes number and current process rank.
00043     GetMPIParams(&processes, &myRank);
00044
00045     // Read data
00046     ifRoot(ReadData(&data, &arraySize, argc, argv));
00047
00048     // Broadcast array dimmensions
00049     broadcastArraySize(&arraySize);
00050
00051     // Scatter data and get the lines assigned to the current process
00052     localData = scatterData(data, arraySize, myRank, processes);
00053
00054     // Check if the matrix meets the criteria
00055     bool result = checkCriteria(localData, arraySize, myRank, processes, &max);
00056
00057     // If it doesn't, abort
00058     if (result == false)
00059     {
00060         MPI_Finalize();
00061         return 0;
00062     }
00063
00064     MPI_Barrier(MPI_COMM_WORLD);
00065
00066     // Calculate the B matrix
00067     int minLocal, minXLocal, minYLocal;
00068     calculateB(localData, arraySize, max, myRank, processes, &minLocal, &minXLocal, &minYLocal);
00069
00070     // Find the min element of B
00071     int min, minX, minY;
00072     findMin(minLocal, minXLocal, minYLocal, &min, &minX, &minY, myRank);
00073
00074     // Print min
00075     ifRoot({
00076         std::cout << "min = " << min << std::endl;
00077     })
00078
00079     // Exit
00080     MPI_Finalize();
00081     return 0;
00082 }
```

6.7 src/networking.cpp File Reference

```
#include "networking.h"
#include "data.h"
#include "mpi.h"
#include "utils.h"
#include <algorithm>
#include <iostream>
```

Include dependency graph for networking.cpp:



Macros

- #define **ROOT_PROCESS** 0

Functions

- void GetMPIParams (int *processesCount, int *rank)

  Returns the number of processes and the rank of the current process.
- int lineToProcess (int line, int totalLines, int processes)

  Returns the number of the process a line of the array will be assigned to.
- int _lineToProcess (int line, int totalLines, int processes)

  [Internal] Line to process assignments based on the modulo of the line.
- std::vector< int > processToLines (int process, int totalLines, int processes)

  Returns a vector with the numbers (base 0) of the lines a process will be assigned.
- int processToLinesCount (int process, int totalLines, int processes)

  The number of lines that will be assigned to a process.
- void broadcastArraySize (int *arraySize)

  Method used to abstract the broadcasting of the array size.
- void calculateDisplsScounts (int *&displs, int *&scounts, int arraySize, int processes, int myRank)

  Calculate the displs and scounts arrays for use with Scatter and Gather methods.
- std::vector< std::vector< int > > scatterData (std::vector< std::vector< int >> data, int arraySize, int my↩ Rank, int processes)

  Scatter the lines of matrix A to the running processes.
- int calculateMax (int maxLocal)

  Find and share the max element of matrix A across all running processes.
- bool checkCriteria (std::vector< std::vector< int >> localData, int arraySize, int myRank, int processes, int *max)

  Checks if matrix A meets the strictly diagonally dominant criteria given the lines assigned to the current process and returns it's max element by reference.
- void calculateB (std::vector< std::vector< int >> localData, int arraySize, int max, int myRank, int processes, int *minLocal, int *minXLocal, int *minYLocal)

  Calculate the B matrix given the lines assigned to the current process and the max element of matrix A.
- void findMin (int minLocal, int minXLocal, int minYLocal, int *min)

  Find and share the min element of matrix B across all running processes.

6.7.1 Detailed Description

Author

Konstantinos Kamaropoulos (kamaropoulos@outlook.com)

Version

0.1

Date

2019-01-05

Copyright

Definition in file networking.cpp.

6.7.2 Function Documentation

6.7.2.1 _lineToProcess()

```
int _lineToProcess (
            int line,
            int totalLines,
            int processes )
```

[Internal] Line to process assignments based on the modulo of the line.

Parameters

line	The number of the line of the array to assign. (base 0)
totalLines	The total number of lines in the array.
processes	The total number of processes. (including the root process)

Returns

int The number of the process the passed line of the array will be assigned to based on the modulo of the line.

Definition at line 73 of file networking.cpp.

```
00074 {
00075     return ((line % (processes)));
00076 }
```

6.7.2.2 broadcastArraySize()

```
void broadcastArraySize (
            int * arraySize )
```

Method used to abstract the broadcasting of the array size.

Parameters

arraySize	[in] The total number of lines in all processes

Definition at line 126 of file networking.cpp.

```
00127 {
00128     MPI_Bcast(arraySize, 1, MPI_INT, 0, MPI_COMM_WORLD);
00129 }
```

6.7.2.3 calculateB()

```
void calculateB (
            std::vector< std::vector< int >> localData,
```

```
            int arraySize,
            int max,
            int myRank,
            int processes,
            int * minLocal,
            int * minXLocal,
            int * minYLocal )
```

Calculate the B matrix given the lines assigned to the current process and the max element of matrix A.

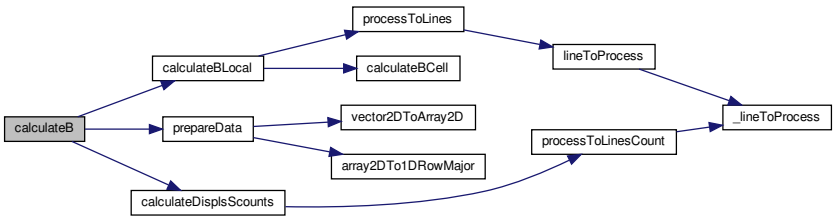
Parameters

localData	[in] A 2D vector with the lines assigned to the current process
arraySize	[in] The total number of lines across all processes
max	[in] The max element of matrix A
myRank	[in] The rank of the current process
processes	[in] The total number of processes
minLocal	[out] The min element only from the lines assigned to the current process
minXLocal	[out] The X coordinate of the local min element
minYLocal	[out] The Y coordinate of the local min element

Definition at line 283 of file networking.cpp.

```
00284 {
00285     std::vector<std::vector<int» localBLines = calculateBLocal(localData, max, arraySize, myRank,
      processes, minLocal, minXLocal, minYLocal);
00286
00287     MPI_Barrier(MPI_COMM_WORLD);
00288
00289     int *localBLinesArr = new int[localBLines.size() * arraySize];
00290     prepareData(localBLines, localBLinesArr);
00291
00292     int *displs, *scounts;
00293     calculateDisplsScounts(displs, scounts, arraySize, processes, myRank);
00294
00295     int *b;
00296     ifRoot({
00297         b = new int[arraySize * arraySize];
00298     });
00299
00300     MPI_Gatherv(localBLinesArr, processToLinesCount(myRank, arraySize, processes) * arraySize,
      MPI_INT, b, scounts, displs, MPI_INT, 0, MPI_COMM_WORLD);
00301
00302     delete[] displs;
00303     delete[] scounts;
00304
00305     delete[] localBLinesArr;
00306
00307     ifRoot({
00308         printB(b, arraySize);
00309     });
00310
00311     delete[] b;
00312 }
```

Here is the call graph for this function:



6.7.2.4 calculateDisplsScounts()

```
void calculateDisplsScounts (
            int *& displs,
            int *& scounts,
            int arraySize,
            int processes,
            int myRank )
```

Calculate the displs and scounts arrays for use with Scatter and Gather methods.

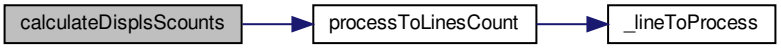
Parameters

displs	[out] A pointer for the displs array to be stored to
scounts	[out] A pointer for the scounts array to be stored to
arraySize	[in] The total number of lines in all processes
processes	[in] The total number of processes
myRank	[in] The rank of the current process

Definition at line 140 of file networking.cpp.

```
00141 {
00142     displs = new int[processes];
00143     scounts = new int[processes];
00144
00145     displs[0] = 0;
00146     scounts[0] = (arraySize * processToLinesCount(0, arraySize, processes));
00147
00148     for (int i = 1; i < processes; i++)
00149     {
00150         if (processToLinesCount(i, arraySize, processes) > 0)
00151         {
00152             displs[i] = displs[i - 1] + scounts[i - 1];
00153             scounts[i] = (arraySize * processToLinesCount(i, arraySize, processes));
00154         }
00155         else
00156         {
00157             displs[i] = 0;
00158             scounts[i] = 0;
00159         }
00160     }
00161 }
```

Here is the call graph for this function:



6.7.2.5 calculateMax()

```
int calculateMax (
            int maxLocal )
```

Find and share the max element of matrix A across all running processes.

Parameters

maxLocal	[in] The max found on the lines assigned to the current process

Returns

int The max element of matrix A

Definition at line 210 of file networking.cpp.

```
00211 {
00212     int max;
00213     MPI_Allreduce(&maxLocal, &max, 1, MPI_INT, MPI_MAX, MPI_COMM_WORLD);
00214     return max;
00215 }
```

6.7.2.6 checkCriteria()

```
bool checkCriteria (
            std::vector< std::vector< int >> localData,
            int arraySize,
            int myRank,
            int processes,
            int * max )
```

Checks if matrix A meets the strictly diagonally dominant criteria given the lines assigned to the current process and returns it's max element by reference.

Parameters

localData	[in] The lines assigned to the current process
arraySize	[in] The total number of lines
myRank	[in] The rank of the current process
processes	[in] The total number of processes
max	[out] The max element of matrix A

Returns

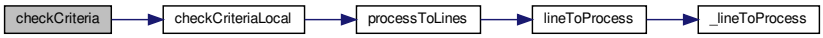
true If matrix A meets the strictly diagonally dominant criteria
false If matrix A does not meet the strictly diagonally dominant criteria

Definition at line 230 of file networking.cpp.

```
00231 {
00232     int maxLocal;
00233     int resultLocal = checkCriteriaLocal(localData, arraySize, myRank, processes, &maxLocal);
00234
00235     int globalResult;
00236
00237     MPI_Reduce(&resultLocal, &globalResult, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
00238
00239     bool abortP = false;
00240
00241     ifRoot({
00242         if (globalResult == processes)
00243         {
00244             abortP = false;
00245             std::cout « "yes" « std::endl;
```

```
00246         }
00247         else
00248         {
00249             abortP = true;
00250             std::cout « "no" « std::endl;
00251         }
00252     })
00253
00254         MPI_Barrier(MPI_COMM_WORLD);
00255
00256     MPI_Bcast(&abortP, 1, MPI_CXX_BOOL, 0, MPI_COMM_WORLD);
00257
00258     if (abortP)
00259     {
00260         return false;
00261     }
00262     else
00263     {
00264         *max = calculateMax(maxLocal);
00265         ifRoot({ std::cout « "max = " « *max « std::endl; });
00266         return true;
00267     }
00268 }
```

Here is the call graph for this function:



6.7.2.7 findMin()

```
void findMin (
              int minLocal,
              int minXLocal,
              int minYLocal,
              int * min )
```

Find and share the min element of matrix B across all running processes.

Parameters

minLocal	[in] The min element only from the lines assigned to the current process
minXLocal	[in] The X coordinate of the local min element
minYLocal	The Y coordinate of the local min element
min	[out] The min elemetn of matrix B

Definition at line 322 of file networking.cpp.

```
00323 {
00324     MPI_Allreduce(&minLocal, min, 1, MPI_INT, MPI_MIN, MPI_COMM_WORLD);
00325 }
```

6.7.2.8 GetMPIParams()

```
void GetMPIParams (
```

```
            int * processesCount,
            int * rank )
```

Returns the number of processes and the rank of the current process.

Parameters

processesCount	The address to which the number of the processes will be written.
rank	The address to which the rank of the current process will be written.

Definition at line 30 of file networking.cpp.

```
00031 {
00032     MPI_Comm_rank(MPI_COMM_WORLD, rank);
00033     MPI_Comm_size(MPI_COMM_WORLD, processesCount);
00034 }
```

6.7.2.9 lineToProcess()

```
int lineToProcess (
            int line,
            int totalLines,
            int processes )
```

Returns the number of the process a line of the array will be assigned to.

Parameters

line	The number of the line of the array to assign. (base 0)
totalLines	The total number of lines in the array.
processes	The total number of processes. (including the root process)

Returns

int The number of the process the passed line of the array will be assigned to.

Definition at line 44 of file networking.cpp.

```
00045 {
00046     // Create a vector holding the number of the process each line will be assigned to.
00047     // The index will be the line number and the data the process number;
00048     std::vector<int> lineAssignments;
00049
00050     // What we want to do here is assign each line to a process based on how many
00051     // there are and then sort the array so all the lines of a process are continuous
00052     // for easier manipulation.
00053
00054     // Fill the vector with the assignments based on the modulo of the line.
00055     for (int _line = 0; _line < totalLines; _line++)
00056         lineAssignments.push_back(_lineToProcess(_line, totalLines, processes));
00057
00058     // Sort the vector so all the process occurances are together.
00059     std::sort(lineAssignments.begin(), lineAssignments.end());
00060
00061     return lineAssignments[line];
00062 }
```

Here is the call graph for this function:

6.7.2.10 processToLines()

```
std::vector<int> processToLines (
            int process,
            int totalLines,
            int processes )
```

Returns a vector with the numbers (base 0) of the lines a process will be assigned.

Parameters

process	The number of the process we want to get the lines it will be assigned.
totalLines	The total number of lines in the array.
processes	The total number of processes. (including the root process)

Returns

std::vector<int> An std::vector<int> object containing the numbers (base 0) of the lines the passed process will be assigned.

Definition at line 87 of file networking.cpp.

```
00088 {
00089     std::vector<int> lines;
00090
00091     for (int line = 0; line < totalLines; line++)
00092     {
00093         if (lineToProcess(line, totalLines, processes) == process)
00094         {
00095             lines.push_back(line);
00096         }
00097     }
00098
00099     return lines;
00100 }
```

Here is the call graph for this function:

6.7.2.11 processToLinesCount()

```
int processToLinesCount (
            int process,
            int totalLines,
            int processes )
```

The number of lines that will be assigned to a process.

Parameters

process	The number of the process
totalLines	The total number of lines in the array.
processes	The total number of processes. (including the root process)

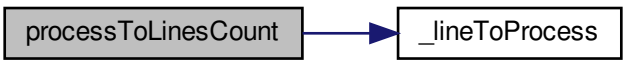
Returns

int The number of lines the passed process will be assigned.

Definition at line 110 of file networking.cpp.

```
00111 {
00112     int linesCount = 0;
00113
00114     for (int i = 0; i < totalLines; i++)
00115         if (_lineToProcess(i, totalLines, processes) == process)
00116             linesCount++;
00117
00118     return linesCount;
00119 }
```

Here is the call graph for this function:



6.7.2.12 scatterData()

```
std::vector<std::vector<int> > scatterData (
            std::vector< std::vector< int >> data,
            int arraySize,
            int myRank,
            int processes )
```

Scatter the lines of matrix A to the running processes.

Parameters

data	[in] A 2D vector with the input data
arraySize	[in] The total number of lines
myRank	[in] The rank of teh current process
processes	[in] The total number of processes

Returns

std::vector<std::vector<int>> The lines assigned to the current process

Definition at line 172 of file networking.cpp.

```
00173 {
00174
00175     // An array to hold the array in row major form
00176     int *dataArray;
00177
00178     // Convert data vector to row-major order array
00179     ifRoot({
00180         dataArray = new int[data.size() * arraySize];
00181         prepareData(data, dataArray);
00182     });
00183
00184     int *displs, *scounts;
00185     calculateDisplsScounts(displs, scounts, arraySize, processes, myRank);
00186
00187     int elements = (arraySize)*processToLinesCount(myRank, arraySize, processes);
00188     int *dataOut = new int[elements];
00189
00190     MPI_Barrier(MPI_COMM_WORLD);
00191
00192     MPI_Scatterv(dataArray, scounts, displs, MPI_INT, dataOut, *scounts, MPI_INT, 0, MPI_COMM_WORLD);
00193
00194     MPI_Barrier(MPI_COMM_WORLD);
00195
00196     delete[] displs;
00197     delete[] scounts;
00198
00199     std::vector<std::vector<int» localData = arrayRowMajorTo2DVector(dataOut,
      processToLinesCount(myRank, arraySize, processes), arraySize);
00200
00201     return localData;
00202 }
```

6.8 networking.cpp

```
00001
00012 #include "networking.h"
00013 #include "data.h"
00014 #include "mpi.h"
00015
00016 #define ROOT_PROCESS 0
00017 #include "utils.h"
00018
00019 #include <algorithm>
00020
00021 #include <iostream>
00022
00030 void GetMPIParams(int *processesCount, int *rank)
00031 {
00032     MPI_Comm_rank(MPI_COMM_WORLD, rank);
00033     MPI_Comm_size(MPI_COMM_WORLD, processesCount);
00034 }
00035
00044 int lineToProcess(int line, int totalLines, int processes)
00045 {
00046     // Create a vector holding the number of the process each line will be assigned to.
00047     // The index will be the line number and the data the process number;
00048     std::vector<int> lineAssignments;
00049
00050     // What we want to do here is assign each line to a process based on how many
00051     // there are and then sort the array so all the lines of a process are continuous
00052     // for easier manipulation.
00053
```

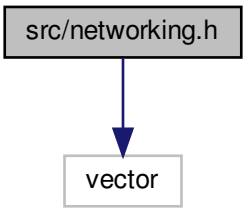
```
00054     // Fill the vector with the assignments based on the modulo of the line.
00055     for (int _line = 0; _line < totalLines; _line++)
00056         lineAssignments.push_back(_lineToProcess(_line, totalLines, processes));
00057 
00058     // Sort the vector so all the process occurances are together.
00059     std::sort(lineAssignments.begin(), lineAssignments.end());
00060 
00061     return lineAssignments[line];
00062 }
00063 
00073 int _lineToProcess(int line, int totalLines, int processes)
00074 {
00075     return ((line % (processes)));
00076 }
00077 
00087 std::vector<int> processToLines(int process, int totalLines, int processes)
00088 {
00089     std::vector<int> lines;
00090 
00091     for (int line = 0; line < totalLines; line++)
00092     {
00093         if (lineToProcess(line, totalLines, processes) == process)
00094         {
00095             lines.push_back(line);
00096         }
00097     }
00098 
00099     return lines;
00100 }
00101 
00110 int processToLinesCount(int process, int totalLines, int processes)
00111 {
00112     int linesCount = 0;
00113 
00114     for (int i = 0; i < totalLines; i++)
00115         if (_lineToProcess(i, totalLines, processes) == process)
00116             linesCount++;
00117 
00118     return linesCount;
00119 }
00120 
00126 void broadcastArraySize(int *arraySize)
00127 {
00128     MPI_Bcast(arraySize, 1, MPI_INT, 0, MPI_COMM_WORLD);
00129 }
00130 
00140 void calculateDisplsScounts(int *&displs, int *&scounts, int arraySize, int processes, int myRank)
00141 {
00142     displs = new int[processes];
00143     scounts = new int[processes];
00144 
00145     displs[0] = 0;
00146     scounts[0] = (arraySize * processToLinesCount(0, arraySize, processes));
00147 
00148     for (int i = 1; i < processes; i++)
00149     {
00150         if (processToLinesCount(i, arraySize, processes) > 0)
00151         {
00152             displs[i] = displs[i - 1] + scounts[i - 1];
00153             scounts[i] = (arraySize * processToLinesCount(i, arraySize, processes));
00154         }
00155         else
00156         {
00157             displs[i] = 0;
00158             scounts[i] = 0;
00159         }
00160     }
00161 }
00162 
00172 std::vector<std::vector<int» scatterData(std::vector<std::vector<int» data, int arraySize, int myRank,
       int processes)
00173 {
00174 
00175     // An array to hold the array in row major form
00176     int *dataArray;
00177 
00178     // Convert data vector to row-major order array
00179     ifRoot({
00180         dataArray = new int[data.size() * arraySize];
00181         prepareData(data, dataArray);
00182     });
00183 
00184     int *displs, *scounts;
00185     calculateDisplsScounts(displs, scounts, arraySize, processes, myRank);
00186 
00187     int elements = (arraySize)*processToLinesCount(myRank, arraySize, processes);
00188     int *dataOut = new int[elements];
```

```
00189
00190     MPI_Barrier(MPI_COMM_WORLD);
00191
00192     MPI_Scatterv(dataArray, scounts, displs, MPI_INT, dataOut, *scounts, MPI_INT, 0, MPI_COMM_WORLD);
00193
00194     MPI_Barrier(MPI_COMM_WORLD);
00195
00196     delete[] displs;
00197     delete[] scounts;
00198
00199     std::vector<std::vector<int> localData = arrayRowMajorTo2DVector(dataOut,
      processToLinesCount(myRank, arraySize, processes), arraySize);
00200
00201     return localData;
00202 }
00203
00210 int calculateMax(int maxLocal)
00211 {
00212     int max;
00213     MPI_Allreduce(&maxLocal, &max, 1, MPI_INT, MPI_MAX, MPI_COMM_WORLD);
00214     return max;
00215 }
00216
00230 bool checkCriteria(std::vector<std::vector<int> localData, int arraySize, int myRank, int processes,
      int *max)
00231 {
00232     int maxLocal;
00233     int resultLocal = checkCriteriaLocal(localData, arraySize, myRank, processes, &maxLocal);
00234
00235     int globalResult;
00236
00237     MPI_Reduce(&resultLocal, &globalResult, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
00238
00239     bool abortP = false;
00240
00241     ifRoot({
00242         if (globalResult == processes)
00243         {
00244             abortP = false;
00245             std::cout « "yes" « std::endl;
00246         }
00247         else
00248         {
00249             abortP = true;
00250             std::cout « "no" « std::endl;
00251         }
00252     })
00253
00254         MPI_Barrier(MPI_COMM_WORLD);
00255
00256     MPI_Bcast(&abortP, 1, MPI_CXX_BOOL, 0, MPI_COMM_WORLD);
00257
00258     if (abortP)
00259     {
00260         return false;
00261     }
00262     else
00263     {
00264         *max = calculateMax(maxLocal);
00265         ifRoot({ std::cout « "max = " « *max « std::endl; });
00266         return true;
00267     }
00268 }
00269
00283 void calculateB(std::vector<std::vector<int> localData, int arraySize, int max, int myRank, int
      processes, int *minLocal, int *minXLocal, int *minYLocal)
00284 {
00285     std::vector<std::vector<int> localBLines = calculateBLocal(localData, max, arraySize, myRank,
      processes, minLocal, minXLocal, minYLocal);
00286
00287     MPI_Barrier(MPI_COMM_WORLD);
00288
00289     int *localBLinesArr = new int[localBLines.size() * arraySize];
00290     prepareData(localBLines, localBLinesArr);
00291
00292     int *displs, *scounts;
00293     calculateDisplsScounts(displs, scounts, arraySize, processes, myRank);
00294
00295     int *b;
00296     ifRoot({
00297         b = new int[arraySize * arraySize];
00298     });
00299
00300     MPI_Gatherv(localBLinesArr, processToLinesCount(myRank, arraySize, processes) * arraySize,
      MPI_INT, b, scounts, displs, MPI_INT, 0, MPI_COMM_WORLD);
00301
00302     delete[] displs;
```

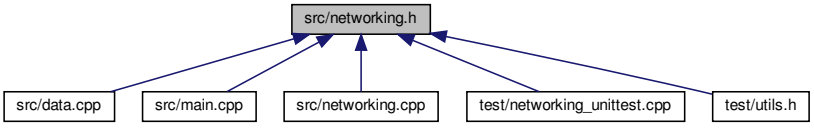
```
00303     delete[] scounts;
00304
00305     delete[] localBLinesArr;
00306
00307     ifRoot({
00308         printB(b, arraySize);
00309     });
00310
00311     delete[] b;
00312 }
00313
00322 void findMin(int minLocal, int minXLocal, int minYLocal, int *min)
00323 {
00324     MPI_Allreduce(&minLocal, min, 1, MPI_INT, MPI_MIN, MPI_COMM_WORLD);
00325 }
```

6.9 src/networking.h File Reference

`#include <vector>`
Include dependency graph for networking.h:



This graph shows which files directly or indirectly include this file:



Functions

- void GetMPIParams (int *processesCount, int *rank)

  Returns the number of processes and the rank of the current process.
- int _lineToProcess (int line, int totalLines, int processes)

  [Internal] Line to process assignments based on the modulo of the line.
- int lineToProcess (int line, int totalLines, int processes)

  Returns the number of the process a line of the array will be assigned to.
- std::vector< int > processToLines (int process, int totalLines, int processes)

Returns a vector with the numbers (base 0) of the lines a process will be assigned.

- int processToLinesCount (int process, int totalLines, int processes)

  The number of lines that will be assigned to a process.
- void broadcastArraySize (int *arraySize)

  Method used to abstract the broadcasting of the array size.
- void **scatterData** (std::vector< std::vector< int >> data)
- void calculateDisplsScounts (int *&, int *&, int, int, int)

  Calculate the displs and scounts arrays for use with Scatter and Gather methods.
- std::vector< std::vector< int > > scatterData (std::vector< std::vector< int >> data, int arraySize, int my↩ Rank, int processes)

  Scatter the lines of matrix A to the running processes.
- bool checkCriteria (std::vector< std::vector< int >> localData, int arraySize, int myRank, int processes, int *maxLocal)

  Checks if matrix A meets the strictly diagonally dominant criteria given the lines assigned to the current process and returns it's max element by reference.
- void calculateB (std::vector< std::vector< int >> localData, int arraySize, int max, int myRank, int processes, int *minLocal, int *minXLocal, int *minYLocal)

  Calculate the B matrix given the lines assigned to the current process and the max element of matrix A.
- void findMin (int minLocal, int minXLocal, int minYLocal, int *min)

  Find and share the min element of matrix B across all running processes.

6.9.1 Detailed Description

Author

Konstantinos Kamaropoulos (kamaropoulos@outlook.com)

Version

0.1

Date

2019-01-05

Copyright

Copyright (c) 2019

Definition in file networking.h.

6.9.2 Function Documentation

6.9.2.1 _lineToProcess()

```
int _lineToProcess (
            int line,
            int totalLines,
            int processes )
```

[Internal] Line to process assignments based on the modulo of the line.

Parameters

line	The number of the line of the array to assign. (base 0)
totalLines	The total number of lines in the array.
processes	The total number of processes. (including the root process)

Returns

int The number of the process the passed line of the array will be assigned to based on the modulo of the line.

Definition at line 73 of file networking.cpp.

```
00074 {
00075     return ((line % (processes)));
00076 }
```

6.9.2.2 broadcastArraySize()

```
void broadcastArraySize (
            int * arraySize )
```

Method used to abstract the broadcasting of the array size.

Parameters

arraySize	[in] The total number of lines in all processes

Definition at line 126 of file networking.cpp.

```
00127 {
00128     MPI_Bcast(arraySize, 1, MPI_INT, 0, MPI_COMM_WORLD);
00129 }
```

6.9.2.3 calculateB()

```
void calculateB (
            std::vector< std::vector< int >> localData,
            int arraySize,
            int max,
            int myRank,
            int processes,
            int * minLocal,
            int * minXLocal,
            int * minYLocal )
```

Calculate the B matrix given the lines assigned to the current process and the max element of matrix A.

Parameters

localData	[in] A 2D vector with the lines assigned to the current process

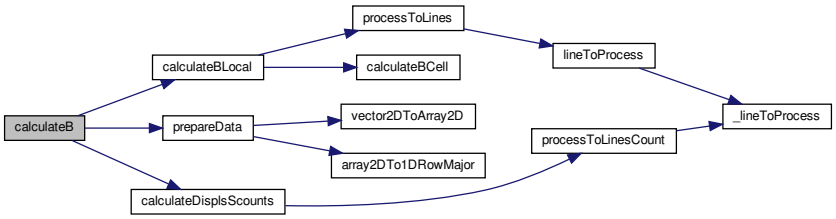
Parameters

arraySize	[in] The total number of lines across all processes
max	[in] The max element of matrix A
myRank	[in] The rank of the current process
processes	[in] The total number of processes
minLocal	[out] The min element only from the lines assigned to the current process
minXLocal	[out] The X coordinate of the local min element
minYLocal	[out] The Y coordinate of the local min element

Definition at line 283 of file networking.cpp.

```
00284 {
00285     std::vector<std::vector<int» localBLines = calculateBLocal(localData, max, arraySize, myRank,
      processes, minLocal, minXLocal, minYLocal);
00286
00287     MPI_Barrier(MPI_COMM_WORLD);
00288
00289     int *localBLinesArr = new int[localBLines.size() * arraySize];
00290     prepareData(localBLines, localBLinesArr);
00291
00292     int *displs, *scounts;
00293     calculateDisplsScounts(displs, scounts, arraySize, processes, myRank);
00294
00295     int *b;
00296     ifRoot({
00297         b = new int[arraySize * arraySize];
00298     });
00299
00300     MPI_Gatherv(localBLinesArr, processToLinesCount(myRank, arraySize, processes) * arraySize,
      MPI_INT, b, scounts, displs, MPI_INT, 0, MPI_COMM_WORLD);
00301
00302     delete[] displs;
00303     delete[] scounts;
00304
00305     delete[] localBLinesArr;
00306
00307     ifRoot({
00308         printB(b, arraySize);
00309     });
00310
00311     delete[] b;
00312 }
```

Here is the call graph for this function:



6.9.2.4 calculateDisplsScounts()

```
void calculateDisplsScounts (
            int *& displs,
            int *& scounts,
```

```
            int arraySize,
            int processes,
            int myRank )
```

Calculate the displs and scounts arrays for use with Scatter and Gather methods.

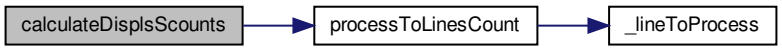
Parameters

displs	[out] A pointer for the displs array to be stored to
scounts	[out] A pointer for the scounts array to be stored to
arraySize	[in] The total number of lines in all processes
processes	[in] The total number of processes
myRank	[in] The rank of the current process

Definition at line 140 of file networking.cpp.

```
00141 {
00142     displs = new int[processes];
00143     scounts = new int[processes];
00144
00145     displs[0] = 0;
00146     scounts[0] = (arraySize * processToLinesCount(0, arraySize, processes));
00147
00148     for (int i = 1; i < processes; i++)
00149     {
00150         if (processToLinesCount(i, arraySize, processes) > 0)
00151         {
00152             displs[i] = displs[i - 1] + scounts[i - 1];
00153             scounts[i] = (arraySize * processToLinesCount(i, arraySize, processes));
00154         }
00155         else
00156         {
00157             displs[i] = 0;
00158             scounts[i] = 0;
00159         }
00160     }
00161 }
```

Here is the call graph for this function:



6.9.2.5 checkCriteria()

```
bool checkCriteria (
            std::vector< std::vector< int >> localData,
            int arraySize,
            int myRank,
            int processes,
            int * max )
```

Checks if matrix A meets the strictly diagonally dominant criteria given the lines assigned to the current process and returns it's max element by reference.

Parameters

localData	[in] The lines assigned to the current process
arraySize	[in] The total number of lines
myRank	[in] The rank of the current process
processes	[in] The total number of processes
max	[out] The max element of matrix A

Returns

true If matrix A meets the strictly diagonally dominant criteria
false If matrix A does not meet the strictly diagonally dominant criteria

Definition at line 230 of file networking.cpp.

```
00231 {
00232     int maxLocal;
00233     int resultLocal = checkCriteriaLocal(localData, arraySize, myRank, processes, &maxLocal);
00234
00235     int globalResult;
00236
00237     MPI_Reduce(&resultLocal, &globalResult, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
00238
00239     bool abortP = false;
00240
00241     ifRoot({
00242         if (globalResult == processes)
00243         {
00244             abortP = false;
00245             std::cout « "yes" « std::endl;
00246         }
00247         else
00248         {
00249             abortP = true;
00250             std::cout « "no" « std::endl;
00251         }
00252     })
00253
00254         MPI_Barrier(MPI_COMM_WORLD);
00255
00256     MPI_Bcast(&abortP, 1, MPI_CXX_BOOL, 0, MPI_COMM_WORLD);
00257
00258     if (abortP)
00259     {
00260         return false;
00261     }
00262     else
00263     {
00264         *max = calculateMax(maxLocal);
00265         ifRoot({ std::cout « "max = " « *max « std::endl; });
00266         return true;
00267     }
00268 }
```

Here is the call graph for this function:

6.9.2.6 findMin()

```
void findMin (
            int minLocal,
            int minXLocal,
            int minYLocal,
            int * min )
```

Find and share the min element of matrix B across all running processes.

Parameters

minLocal	[in] The min element only from the lines assigned to the current process
minXLocal	[in] The X coordinate of the local min element
minYLocal	The Y coordinate of the local min element
min	[out] The min elemetn of matrix B

Definition at line 322 of file networking.cpp.

```
00323 {
00324     MPI_Allreduce(&minLocal, min, 1, MPI_INT, MPI_MIN, MPI_COMM_WORLD);
00325 }
```

6.9.2.7 GetMPIParams()

```
void GetMPIParams (
            int * processesCount,
            int * rank )
```

Returns the number of processes and the rank of the current process.

Parameters

processesCount	The address to which the number of the processes will be written.
rank	The address to which the rank of the current process will be written.

Definition at line 30 of file networking.cpp.

```
00031 {
00032     MPI_Comm_rank(MPI_COMM_WORLD, rank);
00033     MPI_Comm_size(MPI_COMM_WORLD, processesCount);
00034 }
```

6.9.2.8 lineToProcess()

```
int lineToProcess (
            int line,
            int totalLines,
            int processes )
```

Returns the number of the process a line of the array will be assigned to.

Parameters

line	The number of the line of the array to assign. (base 0)
totalLines	The total number of lines in the array.
processes	The total number of processes. (including the root process)

Returns

int The number of the process the passed line of the array will be assigned to.

Definition at line 44 of file networking.cpp.

```
00045 {
00046     // Create a vector holding the number of the process each line will be assigned to.
00047     // The index will be the line number and the data the process number;
00048     std::vector<int> lineAssignments;
00049
00050     // What we want to do here is assign each line to a process based on how many
00051     // there are and then sort the array so all the lines of a process are continuous
00052     // for easier manipulation.
00053
00054     // Fill the vector with the assignments based on the modulo of the line.
00055     for (int _line = 0; _line < totalLines; _line++)
00056         lineAssignments.push_back(_lineToProcess(_line, totalLines, processes));
00057
00058     // Sort the vector so all the process occurances are together.
00059     std::sort(lineAssignments.begin(), lineAssignments.end());
00060
00061     return lineAssignments[line];
00062 }
```

Here is the call graph for this function:

6.9.2.9 processToLines()

```
std::vector<int> processToLines (
            int process,
            int totalLines,
            int processes )
```

Returns a vector with the numbers (base 0) of the lines a process will be assigned.

Parameters

process	The number of the process we want to get the lines it will be assigned.
totalLines	The total number of lines in the array.
processes	The total number of processes. (including the root process)

Returns

std::vector<int> An std::vector<int> object containing the numbers (base 0) of the lines the passed process will be assigned.

Definition at line 87 of file networking.cpp.

```
00088 {
00089     std::vector<int> lines;
00090
00091     for (int line = 0; line < totalLines; line++)
00092     {
00093         if (lineToProcess(line, totalLines, processes) == process)
00094         {
00095             lines.push_back(line);
00096         }
00097     }
00098
00099     return lines;
00100 }
```

Here is the call graph for this function:

6.9.2.10 processToLinesCount()

```
int processToLinesCount (
            int process,
            int totalLines,
            int processes )
```

The number of lines that will be assigned to a process.

Parameters

process	The number of the process
totalLines	The total number of lines in the array.
processes	The total number of processes. (including the root process)

Returns

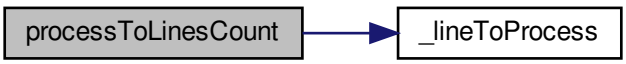
int The number of lines the passed process will be assigned.

Definition at line 110 of file networking.cpp.

```
00111 {
00112     int linesCount = 0;
00113
00114     for (int i = 0; i < totalLines; i++)
00115         if (_lineToProcess(i, totalLines, processes) == process)
00116             linesCount++;
```

```
00117
00118     return linesCount;
00119 }
```

Here is the call graph for this function:



6.9.2.11 scatterData()

```
std::vector<std::vector<int> > scatterData (
              std::vector< std::vector< int >> data,
              int arraySize,
              int myRank,
              int processes )
```

Scatter the lines of matrix A to the running processes.

Parameters

data	[in] A 2D vector with the input data
arraySize	[in] The total number of lines
myRank	[in] The rank of teh current process
processes	[in] The total number of processes

Returns

std::vector<std::vector<int>> The lines assigned to the current process

Definition at line 172 of file networking.cpp.

```
00173 {
00174
00175     // An array to hold the array in row major form
00176     int *dataArray;
00177
00178     // Convert data vector to row-major order array
00179     ifRoot({
00180         dataArray = new int[data.size() * arraySize];
00181         prepareData(data, dataArray);
00182     });
00183
00184     int *displs, *scounts;
00185     calculateDisplsScounts(displs, scounts, arraySize, processes, myRank);
00186
00187     int elements = (arraySize)*processToLinesCount(myRank, arraySize, processes);
00188     int *dataOut = new int[elements];
00189
00190     MPI_Barrier(MPI_COMM_WORLD);
00191
```

```
00192     MPI_Scatterv(dataArray, scounts, displs, MPI_INT, dataOut, *scounts, MPI_INT, 0, MPI_COMM_WORLD);
00193
00194     MPI_Barrier(MPI_COMM_WORLD);
00195
00196     delete[] displs;
00197     delete[] scounts;
00198
00199     std::vector<std::vector<int» localData = arrayRowMajorTo2DVector(dataOut,
      processToLinesCount(myRank, arraySize, processes), arraySize);
00200
00201     return localData;
00202 }
```

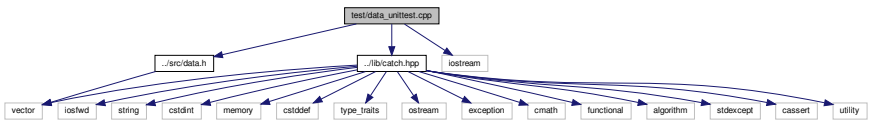
6.10 networking.h

```
00001
00012 #include <vector>
00013
00014 void GetMPIParams(int *processesCount, int *rank);
00015
00016 int _lineToProcess(int line, int totalLines, int processes);
00017
00018 int lineToProcess(int line, int totalLines, int processes);
00019
00020 std::vector<int> processToLines(int process, int totalLines, int processes);
00021
00022 int processToLinesCount(int process, int totalLines, int processes);
00023
00024 void broadcastArraySize(int *arraySize);
00025
00026 void scatterData(std::vector<std::vector<int» data);
00027
00028 void calculateDisplsScounts(int *&, int *&, int, int, int);
00029
00030 std::vector<std::vector<int» scatterData(std::vector<std::vector<int» data, int arraySize, int myRank,
      int processes);
00031
00032 bool checkCriteria(std::vector<std::vector<int» localData, int arraySize, int myRank, int processes,
      int *maxLocal);
00033
00034 void calculateB(std::vector<std::vector<int» localData, int arraySize, int max, int myRank, int
      processes, int *minLocal, int *minXLocal, int *minYLocal);
00035
00036 void findMin(int minLocal, int minXLocal, int minYLocal, int *min);
```

6.11 test/data_unittest.cpp File Reference

```
#include "../lib/catch.hpp"
#include "../src/data.h"
#include <iostream>
```

Include dependency graph for data_unittest.cpp:



Functions

- **TEST_CASE** ("Prepare Data for Gathering: 1x4", "[prepareData]")
- **TEST_CASE** ("Convert 2D Array to 1D Array in Row-Major Order", "[array2DTo1DRowMajor]")
- **TEST_CASE** ("Prepare Data for Gathering: 2x4", "[prepareData]")
- **TEST_CASE** ("Check criteria for local data: true", "[checkCriteriaLocal]")
- **TEST_CASE** ("Check criteria for local data: false", "[checkCriteriaLocal]")
- **TEST_CASE** ("Calculate Local B Lines: Process 0, 4 Lines, 4 Processes", "[calculateBLocal]")

6.11.1 Detailed Description

Author

Konstantinos Kamaropoulos (kamaropoulos@outlook.com)

Version

0.1

Date

2019-01-05

Copyright

Definition in file data_unittest.cpp.

6.12 data_unittest.cpp

```
00001
00012 #include "../lib/catch.hpp"
00013
00014 #include "../src/data.h"
00015
00016 #include <iostream>
00017
00018 TEST_CASE("Prepare Data for Gathering: 1x4", "[prepareData]")
00019 {
00020     int elementsPerLine = 4;
00021     std::vector<std::vector<int» vec;
00022     std::vector<int> l0 = {1, 2, 3, 4};
00023     vec.push_back(l0);
00024
00025     int *arr = new int[vec.size() * 4];
00026     prepareData(vec, arr);
00027
00028     CHECK(arr[0] == vec[0][0]);
00029     CHECK(arr[1] == vec[0][1]);
00030     CHECK(arr[2] == vec[0][2]);
00031     CHECK(arr[3] == vec[0][3]);
00032 }
00033
00034 TEST_CASE("Convert 2D Array to 1D Array in Row-Major Order", "[array2DTo1DRowMajor]")
00035 {
00036     int **arr2D = new int *[1];
00037     arr2D[0] = new int[4];
00038
00039     arr2D[0][0] = 1;
00040     arr2D[0][1] = 2;
00041     arr2D[0][2] = 3;
00042     arr2D[0][3] = 4;
00043
00044     int *arr1D = array2DTo1DRowMajor(arr2D, 1, 4);
00045
00046     CHECK(arr1D[0] == arr2D[0][0]);
00047     CHECK(arr1D[1] == arr2D[0][1]);
00048     CHECK(arr1D[2] == arr2D[0][2]);
00049     CHECK(arr1D[3] == arr2D[0][3]);
00050 }
00051
00052 TEST_CASE("Prepare Data for Gathering: 2x4", "[prepareData]")
00053 {
00054     int elementsPerLine = 4;
00055     std::vector<std::vector<int» vec;
00056     std::vector<int> l0 = {1, 2, 3, 4};
00057     std::vector<int> l1 = {5, 6, 7, 8};
00058     vec.push_back(l0);
```

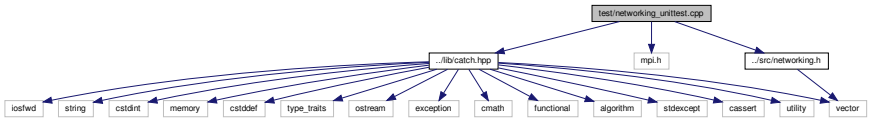
```
00059     vec.push_back(11);
00060
00061     int *arr = new int[vec.size() * 4];
00062     prepareData(vec, arr);
00063
00064     CHECK(arr[0] == vec[0][0]);
00065     CHECK(arr[1] == vec[0][1]);
00066     CHECK(arr[2] == vec[0][2]);
00067     CHECK(arr[3] == vec[0][3]);
00068     CHECK(arr[4] == vec[1][0]);
00069     CHECK(arr[5] == vec[1][1]);
00070     CHECK(arr[6] == vec[1][2]);
00071     CHECK(arr[7] == vec[1][3]);
00072 }
00073
00074 TEST_CASE("Check criteria for local data: true", "[checkCriteriaLocal]")
00075 {
00076     std::vector<std::vector<int» localData;
00077     int arraySize = 4;
00078
00079     std::vector<int> d0 = {12, 3, 4, 1};
00080     std::vector<int> d1 = {3, 46, 1, 1};
00081     std::vector<int> d2 = {4, 1, 38, 2};
00082     std::vector<int> d3 = {3, 2, 1, 25};
00083     localData.push_back(d0);
00084     localData.push_back(d1);
00085     localData.push_back(d2);
00086     localData.push_back(d3);
00087
00088     int maxLocal;
00089
00090     bool resultLocal = checkCriteriaLocal(localData, arraySize, 0, 1, &maxLocal);
00091
00092     CHECK(resultLocal == true);
00093     CHECK(maxLocal == 46);
00094 }
00095
00096 TEST_CASE("Check criteria for local data: false", "[checkCriteriaLocal]")
00097 {
00098     std::vector<std::vector<int» localData;
00099     int arraySize = 4;
00100
00101     std::vector<int> d0 = {12, 3, 4, 1};
00102     std::vector<int> d1 = {3, 1, 1, 1};
00103     std::vector<int> d2 = {4, 1, 38, 2};
00104     std::vector<int> d3 = {3, 2, 1, 25};
00105     localData.push_back(d0);
00106     localData.push_back(d1);
00107     localData.push_back(d2);
00108     localData.push_back(d3);
00109
00110     int maxLocal;
00111
00112     bool resultLocal = checkCriteriaLocal(localData, arraySize, 0, 1, &maxLocal);
00113
00114     CHECK(resultLocal == false);
00115     CHECK(maxLocal == 38);
00116 }
00117
00118 TEST_CASE("Calculate Local B Lines: Process 0, 4 Lines, 4 Processes", "[calculateBLocal]")
00119 {
00120     std::vector<std::vector<int» localData;
00121     int arraySize = 4, myRank = 0, max = 12, processes = 4;
00122
00123     std::vector<int> d0 = {8, 1, 2, 2};
00124     localData.push_back(d0);
00125
00126     int min, minX, minY;
00127
00128     std::vector<std::vector<int» localB = calculateBLocal(localData, max, arraySize, myRank,
      processes, &min, &minX, &minY);
00129
00130     std::vector<std::vector<int» controlB;
00131
00132     std::vector<int> c0 = {12, 11, 10, 10};
00133     controlB.push_back(c0);
00134
00135     CHECK(localB[0][0] == controlB[0][0]);
00136     CHECK(localB[0][1] == controlB[0][1]);
00137     CHECK(localB[0][2] == controlB[0][2]);
00138     CHECK(localB[0][3] == controlB[0][3]);
00139
00140     CHECK(min == 10);
00141     CHECK(minX == 0);
00142     CHECK(minY == 2);
00143 }
00144
```

```
00145 TEST_CASE("Calculate Local B Lines: Process 1, 4 Lines, 4 Processes", "[calculateBLocal]")
00146 {
00147     std::vector<std::vector<int> localData;
00148     int arraySize = 4, myRank = 1, max = 12, processes = 4;
00149 
00150     std::vector<int> d0 = {2, 12, 4, 1};
00151     localData.push_back(d0);
00152 
00153     int min, minX, minY;
00154 
00155     std::vector<std::vector<int> localB = calculateBLocal(localData, max, arraySize, myRank,
      processes, &min, &minX, &minY);
00156 
00157     std::vector<std::vector<int> controlB;
00158 
00159     std::vector<int> c0 = {10, 12, 8, 11};
00160     controlB.push_back(c0);
00161 
00162     CHECK(localB[0][0] == controlB[0][0]);
00163     CHECK(localB[0][1] == controlB[0][1]);
00164     CHECK(localB[0][2] == controlB[0][2]);
00165     CHECK(localB[0][3] == controlB[0][3]);
00166 
00167     CHECK(min == 8);
00168     CHECK(minX == 1);
00169     CHECK(minY == 2);
00170 }
```

6.13 test/networking_unittest.cpp File Reference

```
#include "../lib/catch.hpp"
#include "mpi.h"
#include "../src/networking.h"
```

Include dependency graph for networking_unittest.cpp:



Functions

- TEST_CASE ("[Internal] Line to Process Assignments: (p == n), ((n mod p) == 0)", "[ilineToProcess]")

  Unit test for method _lineToProcess with 4 lines and 4 processes (p == n), ((n mod p) == 0)
- TEST_CASE ("[Internal] Line to Process Assignments: (p < n), ((n mod p) == 0)", "[ilineToProcess]")

  Unit test for method _lineToProcess with 8 lines and 4 processes (p < n), ((n mod p) == 0)
- TEST_CASE ("[Internal] Line to Process Assignments: (p < n), ((n mod p) != 0)", "[ilineToProcess]")

  Unit test for method _lineToProcess with 7 lines and 4 processes (p < n), ((n mod p) != 0)
- TEST_CASE ("[Internal] Line to Process Assignments: (p > n), ((n mod p) == 0)", "[ilineToProcess]")

  Unit test for method _lineToProcess with 4 lines and 8 processes (p > n), ((n mod p) == 0)
- TEST_CASE ("[Internal] Line to Process Assignments: (p > n), ((n mod p) != 0)", "[ilineToProcess]")

  Unit test for method _lineToProcess with 5 lines and 8 processes (p > n), ((n mod p) != 0)
- TEST_CASE ("Line to Process Assignments: (p == n), ((n mod p) == 0)", "[lineToProcess]")

  Unit test for method lineToProcess with 4 lines and 4 processes (p == n), ((n mod p) == 0)
- TEST_CASE ("Line to Process Assignments: (p < n), ((n mod p) == 0)", "[lineToProcess]")

  Unit test for method lineToProcess with 8 lines and 4 processes (p < n), ((n mod p) == 0)
- TEST_CASE ("Line to Process Assignments: (p < n), ((n mod p) != 0)", "[lineToProcess]")

  Unit test for method lineToProcess with 7 lines and 4 processes (p < n), ((n mod p) != 0)

- TEST_CASE ("Line to Process Assignments: (p > n), ((n mod p) == 0)", "[lineToProcess]")
  Unit test for method lineToProcess with 4 lines and 8 processes (p > n), ((n mod p) == 0)
- TEST_CASE ("Line to Process Assignments: (p > n), ((n mod p) != 0)", "[lineToProcess]")
  Unit test for method lineToProcess with 5 lines and 8 processes (p > n), ((n mod p) != 0)
- TEST_CASE ("Lines assigned to a process: (p == n), ((n mod p) == 0)", "[processToLines]")
  Unit test for method processToLines with 4 lines and 4 processes (p == n), ((n mod p) == 0)
- TEST_CASE ("Lines assigned to a process: (p < n), ((n mod p) == 0)", "[processToLines]")
  Unit test for method processToLines with 8 lines and 4 processes (p < n), ((n mod p) == 0)
- TEST_CASE ("Lines assigned to a process: (p < n), ((n mod p) != 0)", "[processToLines]")
  Unit test for method processToLines with 7 lines and 4 processes (p < n), ((n mod p) != 0)
- TEST_CASE ("Lines assigned to a process: (p > n), ((n mod p) == 0)", "[processToLines]")
  Unit test for method processToLines with 4 lines and 8 processes (p > n), ((n mod p) == 0)
- TEST_CASE ("Lines assigned to a process: (p > n), ((n mod p) != 0)", "[processToLines]")
  Unit test for method processToLines with 5 lines and 8 processes (p > n), ((n mod p) != 0)
- TEST_CASE ("Number of lines assigned to a process: (p == n), ((n mod p) == 0)", "[processToLinesCount]")
  Unit test for method processToLinesCount with 4 lines and 4 processes (p == n), ((n mod p) == 0)
- TEST_CASE ("Number of lines assigned to a process: (p < n), ((n mod p) == 0)", "[processToLinesCount]")
  Unit test for method processToLinesCount with 8 lines and 4 processes (p < n), ((n mod p) == 0)
- TEST_CASE ("Number of lines assigned to a process: (p < n), ((n mod p) != 0)", "[processToLinesCount]")
  Unit test for method processToLinesCount with 7 lines and 4 processes (p < n), ((n mod p) != 0)
- TEST_CASE ("Number of lines assigned to a process: (p > n), ((n mod p) == 0)", "[processToLinesCount]")
  Unit test for method processToLinesCount with 4 lines and 8 processes (p > n), ((n mod p) == 0)
- TEST_CASE ("Number of lines assigned to a process: (p > n), ((n mod p) != 0)", "[processToLinesCount]")
  Unit test for method processToLinesCount with 5 lines and 8 processes (p > n), ((n mod p) != 0)
- TEST_CASE ("Calculate didpls and scounts for MPI_Scatterv: (p == n), ((n mod p) == 0)", "[calculateDisplsScounts]")
  Unit test for method calculateDisplsScounts with 4 lines and 4 processes (p == n), ((n mod p) == 0)
- TEST_CASE ("Calculate didpls and scounts for MPI_Scatterv: (p < n), ((n mod p) == 0)", "[calculateDisplsScounts]")
  Unit test for method calculateDisplsScounts with 8 lines and 4 processes (p < n), ((n mod p) == 0)
- TEST_CASE ("Calculate didpls and scounts for MPI_Scatterv: (p < n), ((n mod p) != 0)", "[calculateDisplsScounts]")
  Unit test for method calculateDisplsScounts with 7 lines and 4 processes (p < n), ((n mod p) != 0)
- TEST_CASE ("Calculate didpls and scounts for MPI_Scatterv: (p > n), ((n mod p) == 0)", "[calculateDisplsScounts]")
  Unit test for method calculateDisplsScounts with 4 lines and 8 processes (p > n), ((n mod p) == 0)
- TEST_CASE ("Calculate didpls and scounts for MPI_Scatterv: (p > n), ((n mod p) != 0)", "[calculateDisplsScounts]")
  Unit test for method calculateDisplsScounts with 5 lines and 8 processes (p > n), ((n mod p) != 0)

6.13.1 Detailed Description

Author

Konstantinos Kamaropoulos (kamaropoulos@outlook.com)

Version

0.1

Date

2019-01-05

Copyright

Copyright (c) 2019

Definition in file networking_unittest.cpp.

6.13.2 Function Documentation

6.13.2.1 TEST_CASE() [1/25]

```
TEST_CASE (
              " Line to Process Assignments:     (p == n)[Internal],
              ((n mod p)==0)" ,
              "" [ilineToProcess] )
```

Unit test for method _lineToProcess with 4 lines and 4 processes (p == n), ((n mod p) == 0)

Definition at line 24 of file networking_unittest.cpp.

```
00024                                                : (p == n), ((n mod p) == 0)", "[ilineToProcess]")
00025 {
00026     CHECK(_lineToProcess(0, 4, 4) == 0);
00027     CHECK(_lineToProcess(1, 4, 4) == 1);
00028     CHECK(_lineToProcess(2, 4, 4) == 2);
00029     CHECK(_lineToProcess(3, 4, 4) == 3);
00030 }
```

Here is the call graph for this function:

6.13.2.2 TEST_CASE() [2/25]

```
TEST_CASE (
              " Line to Process Assignments:     (p < n)[Internal],
              ((n mod p)==0)" ,
              "" [ilineToProcess] )
```

Unit test for method _lineToProcess with 8 lines and 4 processes (p < n), ((n mod p) == 0)

Definition at line 36 of file networking_unittest.cpp.

```
00036                                                : (p < n), ((n mod p) == 0)", "[ilineToProcess]")
00037 {
00038     CHECK(_lineToProcess(0, 8, 4) == 0);
00039     CHECK(_lineToProcess(1, 8, 4) == 1);
00040     CHECK(_lineToProcess(2, 8, 4) == 2);
00041     CHECK(_lineToProcess(3, 8, 4) == 3);
00042     CHECK(_lineToProcess(4, 8, 4) == 0);
00043     CHECK(_lineToProcess(5, 8, 4) == 1);
00044     CHECK(_lineToProcess(6, 8, 4) == 2);
00045     CHECK(_lineToProcess(7, 8, 4) == 3);
00046 }
```

Here is the call graph for this function:

6.13.2.3 TEST_CASE() [3/25]

```
TEST_CASE (
             " Line to Process Assignments:    (p < n)[Internal],
             ((n mod p) !=0)" ,
             "" [ilineToProcess] )
```

Unit test for method _lineToProcess with 7 lines and 4 processes (p < n), ((n mod p) != 0)

Definition at line 52 of file networking_unittest.cpp.

```
00052                                                 : (p < n), ((n mod p) != 0)", "[ilineToProcess]")
00053 {
00054     CHECK(_lineToProcess(0, 7, 4) == 0);
00055     CHECK(_lineToProcess(1, 7, 4) == 1);
00056     CHECK(_lineToProcess(2, 7, 4) == 2);
00057     CHECK(_lineToProcess(3, 7, 4) == 3);
00058     CHECK(_lineToProcess(4, 7, 4) == 0);
00059     CHECK(_lineToProcess(5, 7, 4) == 1);
00060     CHECK(_lineToProcess(6, 7, 4) == 2);
00061 }
```

Here is the call graph for this function:

6.13.2.4 TEST_CASE() [4/25]

```
TEST_CASE (
              " Line to Process Assignments:     (p > n)[Internal],
              ((n mod p)==0)" ,
              "" [ilineToProcess] )
```

Unit test for method _lineToProcess with 4 lines and 8 processes (p > n), ((n mod p) == 0)

Definition at line 67 of file networking_unittest.cpp.

```
00067                                                : (p > n), ((n mod p) == 0)", "[ilineToProcess]")
00068 {
00069     CHECK(_lineToProcess(0, 4, 8) == 0);
00070     CHECK(_lineToProcess(1, 4, 8) == 1);
00071     CHECK(_lineToProcess(2, 4, 8) == 2);
00072     CHECK(_lineToProcess(3, 4, 8) == 3);
00073 }
```

Here is the call graph for this function:

6.13.2.5 TEST_CASE() [5/25]

```
TEST_CASE (
              " Line to Process Assignments:     (p > n)[Internal],
              ((n mod p) !=0)" ,
              "" [ilineToProcess] )
```

Unit test for method _lineToProcess with 5 lines and 8 processes (p > n), ((n mod p) != 0)

Definition at line 79 of file networking_unittest.cpp.

```
00079                                                : (p > n), ((n mod p) != 0)", "[ilineToProcess]")
00080 {
00081     CHECK(_lineToProcess(0, 5, 8) == 0);
00082     CHECK(_lineToProcess(1, 5, 8) == 1);
00083     CHECK(_lineToProcess(2, 5, 8) == 2);
00084     CHECK(_lineToProcess(3, 5, 8) == 3);
00085     CHECK(_lineToProcess(4, 5, 8) == 4);
00086 }
```

Here is the call graph for this function:

6.13.2.6 TEST_CASE() [6/25]

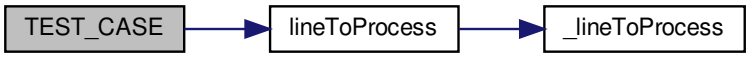
```
TEST_CASE (
             "Line to Process Assignments:    p == n,
             ((n mod p)==0)" ,
             "" [lineToProcess] )
```

Unit test for method lineToProcess with 4 lines and 4 processes (p == n), ((n mod p) == 0)

Definition at line 92 of file networking_unittest.cpp.

```
00092                                     : (p == n), ((n mod p) == 0)", "[lineToProcess]")
00093 {
00094     CHECK(lineToProcess(0, 4, 4) == 0);
00095     CHECK(lineToProcess(1, 4, 4) == 1);
00096     CHECK(lineToProcess(2, 4, 4) == 2);
00097     CHECK(lineToProcess(3, 4, 4) == 3);
00098 }
```

Here is the call graph for this function:



6.13.2.7 TEST_CASE() [7/25]

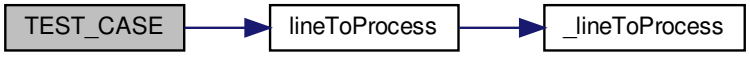
```
TEST_CASE (
             "Line to Process Assignments:    p < n,
             ((n mod p)==0)" ,
             "" [lineToProcess] )
```

Unit test for method lineToProcess with 8 lines and 4 processes (p < n), ((n mod p) == 0)

Definition at line 104 of file networking_unittest.cpp.

```
00104                                     : (p < n), ((n mod p) == 0)", "[lineToProcess]")
00105 {
00106     CHECK(lineToProcess(0, 8, 4) == 0);
00107     CHECK(lineToProcess(1, 8, 4) == 0);
00108     CHECK(lineToProcess(2, 8, 4) == 1);
00109     CHECK(lineToProcess(3, 8, 4) == 1);
00110     CHECK(lineToProcess(4, 8, 4) == 2);
00111     CHECK(lineToProcess(5, 8, 4) == 2);
00112     CHECK(lineToProcess(6, 8, 4) == 3);
00113     CHECK(lineToProcess(7, 8, 4) == 3);
00114 }
```

Here is the call graph for this function:



6.13.2.8 TEST_CASE() [8/25]

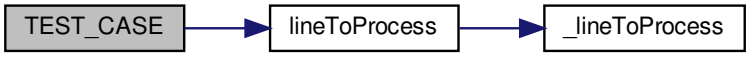
```
TEST_CASE (
              "Line to Process Assignments:    p < n,
              ((n mod p) !=0)" ,
              "" [lineToProcess] )
```

Unit test for method lineToProcess with 7 lines and 4 processes (p < n), ((n mod p) != 0)

Definition at line 120 of file networking_unittest.cpp.

```
00120                                     : (p < n), ((n mod p) != 0)", "[lineToProcess]")
00121 {
00122     CHECK(lineToProcess(0, 7, 4) == 0);
00123     CHECK(lineToProcess(1, 7, 4) == 0);
00124     CHECK(lineToProcess(2, 7, 4) == 1);
00125     CHECK(lineToProcess(3, 7, 4) == 1);
00126     CHECK(lineToProcess(4, 7, 4) == 2);
00127     CHECK(lineToProcess(5, 7, 4) == 2);
00128     CHECK(lineToProcess(6, 7, 4) == 3);
00129 }
```

Here is the call graph for this function:



6.13.2.9 TEST_CASE() [9/25]

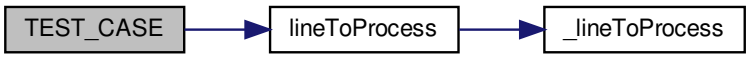
```
TEST_CASE (
              "Line to Process Assignments:    p > n,
              ((n mod p)==0)" ,
              "" [lineToProcess] )
```

Unit test for method lineToProcess with 4 lines and 8 processes (p > n), ((n mod p) == 0)

Definition at line 135 of file networking_unittest.cpp.

```
00135                                     : (p > n), ((n mod p) == 0)", "[lineToProcess]")
00136 {
00137     CHECK(lineToProcess(0, 4, 8) == 0);
00138     CHECK(lineToProcess(1, 4, 8) == 1);
00139     CHECK(lineToProcess(2, 4, 8) == 2);
00140     CHECK(lineToProcess(3, 4, 8) == 3);
00141 }
```

Here is the call graph for this function:



6.13.2.10 TEST_CASE() [10/25]

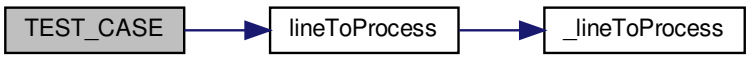
```
TEST_CASE (
            "Line to Process Assignments:    p > n,
            ((n mod p) !=0)" ,
            "" [lineToProcess] )
```

Unit test for method lineToProcess with 5 lines and 8 processes (p > n), ((n mod p) != 0)

Definition at line 147 of file networking_unittest.cpp.

```
00147                                    : (p > n), ((n mod p) != 0)", "[lineToProcess]")
00148 {
00149     CHECK(lineToProcess(0, 5, 8) == 0);
00150     CHECK(lineToProcess(1, 5, 8) == 1);
00151     CHECK(lineToProcess(2, 5, 8) == 2);
00152     CHECK(lineToProcess(3, 5, 8) == 3);
00153     CHECK(lineToProcess(4, 5, 8) == 4);
00154 }
```

Here is the call graph for this function:



6.13.2.11 TEST_CASE() [11/25]

```
TEST_CASE (
            "Lines assigned to a process:    p == n,
            ((n mod p)==0)" ,
            "" [processToLines] )
```

Unit test for method processToLines with 4 lines and 4 processes (p == n), ((n mod p) == 0)

Definition at line 160 of file networking_unittest.cpp.

```
00160                                    : (p == n), ((n mod p) == 0)", "[processToLines]")
00161 {
00162     CHECK(processToLines(0, 4, 4).size() == 1);
00163     CHECK(processToLines(1, 4, 4).size() == 1);
00164     CHECK(processToLines(2, 4, 4).size() == 1);
00165     CHECK(processToLines(3, 4, 4).size() == 1);
00166
00167     CHECK(processToLines(0, 4, 4)[0] == 0);
00168     CHECK(processToLines(1, 4, 4)[0] == 1);
00169     CHECK(processToLines(2, 4, 4)[0] == 2);
00170     CHECK(processToLines(3, 4, 4)[0] == 3);
00171 }
```

Here is the call graph for this function:

6.13.2.12 TEST_CASE() [12/25]

```
TEST_CASE (
            "Lines assigned to a process:    p < n,
            ((n mod p)==0)" ,
            "" [processToLines] )
```

Unit test for method processToLines with 8 lines and 4 processes (p < n), ((n mod p) == 0)

Definition at line 177 of file networking_unittest.cpp.

```
00177                                       : (p < n), ((n mod p) == 0)", "[processToLines]")
00178 {
00179     CHECK(processToLines(0, 8, 4).size() == 2);
00180     CHECK(processToLines(1, 8, 4).size() == 2);
00181     CHECK(processToLines(2, 8, 4).size() == 2);
00182     CHECK(processToLines(3, 8, 4).size() == 2);
00183     CHECK(processToLines(0, 8, 4)[0] == 0);
00184     CHECK(processToLines(0, 8, 4)[1] == 1);
00185     CHECK(processToLines(1, 8, 4)[0] == 2);
00186     CHECK(processToLines(1, 8, 4)[1] == 3);
00187     CHECK(processToLines(2, 8, 4)[0] == 4);
00188     CHECK(processToLines(2, 8, 4)[1] == 5);
00189     CHECK(processToLines(3, 8, 4)[0] == 6);
00190     CHECK(processToLines(3, 8, 4)[1] == 7);
00191 }
```

Here is the call graph for this function:

6.13.2.13 TEST_CASE() [13/25]

```
TEST_CASE (
            "Lines assigned to a process:    p < n,
            ((n mod p) !=0)" ,
            "" [processToLines] )
```

Unit test for method processToLines with 7 lines and 4 processes (p < n), ((n mod p) != 0)

Definition at line 197 of file networking_unittest.cpp.

```
00197                                       : (p < n), ((n mod p) != 0)", "[processToLines]")
00198 {
00199     CHECK(processToLines(0, 7, 4).size() == 2);
00200     CHECK(processToLines(1, 7, 4).size() == 2);
00201     CHECK(processToLines(2, 7, 4).size() == 2);
00202     CHECK(processToLines(3, 7, 4).size() == 1);
00203     CHECK(processToLines(0, 7, 4)[0] == 0);
00204     CHECK(processToLines(0, 7, 4)[1] == 1);
00205     CHECK(processToLines(1, 7, 4)[0] == 2);
00206     CHECK(processToLines(1, 7, 4)[1] == 3);
00207     CHECK(processToLines(2, 7, 4)[0] == 4);
00208     CHECK(processToLines(2, 7, 4)[1] == 5);
00209     CHECK(processToLines(3, 7, 4)[0] == 6);
00210 }
```

Here is the call graph for this function:

6.13.2.14 TEST_CASE() [14/25]

```
TEST_CASE (
            "Lines assigned to a process:    p > n,
            ((n mod p)==0)" ,
            "" [processToLines] )
```

Unit test for method processToLines with 4 lines and 8 processes (p > n), ((n mod p) == 0)

Definition at line 216 of file networking_unittest.cpp.

```
00216                                     : (p > n), ((n mod p) == 0)", "[processToLines]")
00217 {
00218     CHECK(processToLines(0, 4, 8).size() == 1);
00219     CHECK(processToLines(1, 4, 8).size() == 1);
00220     CHECK(processToLines(2, 4, 8).size() == 1);
00221     CHECK(processToLines(3, 4, 8).size() == 1);
00222     CHECK(processToLines(4, 4, 8).size() == 0);
00223     CHECK(processToLines(5, 4, 8).size() == 0);
00224     CHECK(processToLines(6, 4, 8).size() == 0);
00225     CHECK(processToLines(7, 4, 8).size() == 0);
00226     CHECK(processToLines(0, 4, 8)[0] == 0);
00227     CHECK(processToLines(1, 4, 8)[0] == 1);
00228     CHECK(processToLines(2, 4, 8)[0] == 2);
00229     CHECK(processToLines(3, 4, 8)[0] == 3);
00230 }
```

Here is the call graph for this function:

6.13.2.15 TEST_CASE() [15/25]

```
TEST_CASE (
            "Lines assigned to a process:    p > n,
            ((n mod p) !=0)" ,
            "" [processToLines] )
```

Unit test for method processToLines with 5 lines and 8 processes (p > n), ((n mod p) != 0)

Definition at line 236 of file networking_unittest.cpp.

```
00236                                       : (p > n), ((n mod p) != 0)", "[processToLines]")
00237 {
00238     CHECK(processToLines(0, 5, 8).size() == 1);
00239     CHECK(processToLines(1, 5, 8).size() == 1);
00240     CHECK(processToLines(2, 5, 8).size() == 1);
00241     CHECK(processToLines(3, 5, 8).size() == 1);
00242     CHECK(processToLines(4, 5, 8).size() == 1);
00243     CHECK(processToLines(5, 5, 8).size() == 0);
00244     CHECK(processToLines(6, 5, 8).size() == 0);
00245     CHECK(processToLines(7, 5, 8).size() == 0);
00246     CHECK(processToLines(0, 5, 8)[0] == 0);
00247     CHECK(processToLines(1, 5, 8)[0] == 1);
00248     CHECK(processToLines(2, 5, 8)[0] == 2);
00249     CHECK(processToLines(3, 5, 8)[0] == 3);
00250     CHECK(processToLines(4, 5, 8)[0] == 4);
00251 }
```

Here is the call graph for this function:

6.13.2.16 TEST_CASE() [16/25]

```
TEST_CASE (
             "Number of lines assigned to a process:    p == n,
             ((n mod p)==0)" ,
             "" [processToLinesCount] )
```

Unit test for method processToLinesCount with 4 lines and 4 processes (p == n), ((n mod p) == 0)

Definition at line 257 of file networking_unittest.cpp.

```
00257                                       : (p == n), ((n mod p) == 0)",
      "[processToLinesCount]")
00258 {
00259     CHECK(processToLines(0, 4, 4).size() == 1);
00260     CHECK(processToLines(1, 4, 4).size() == 1);
00261     CHECK(processToLines(2, 4, 4).size() == 1);
00262     CHECK(processToLines(3, 4, 4).size() == 1);
00263     CHECK(processToLines(0, 4, 4)[0] == 0);
00264     CHECK(processToLines(1, 4, 4)[0] == 1);
00265     CHECK(processToLines(2, 4, 4)[0] == 2);
00266     CHECK(processToLines(3, 4, 4)[0] == 3);
00267 }
```

Here is the call graph for this function:

6.13.2.17 TEST_CASE() [17/25]

```
TEST_CASE (
            "Number of lines assigned to a process:    p < n,
            ((n mod p)==0)" ,
            "" [processToLinesCount] )
```

Unit test for method processToLinesCount with 8 lines and 4 processes (p < n), ((n mod p) == 0)

Definition at line 273 of file networking_unittest.cpp.

```
00273                                                 : (p < n), ((n mod p) == 0)", "[processToLinesCount]")
00274 {
00275     CHECK(processToLines(0, 8, 4).size() == 2);
00276     CHECK(processToLines(1, 8, 4).size() == 2);
00277     CHECK(processToLines(2, 8, 4).size() == 2);
00278     CHECK(processToLines(3, 8, 4).size() == 2);
00279     CHECK(processToLines(0, 8, 4)[0] == 0);
00280     CHECK(processToLines(0, 8, 4)[1] == 1);
00281     CHECK(processToLines(1, 8, 4)[0] == 2);
00282     CHECK(processToLines(1, 8, 4)[1] == 3);
00283     CHECK(processToLines(2, 8, 4)[0] == 4);
00284     CHECK(processToLines(2, 8, 4)[1] == 5);
00285     CHECK(processToLines(3, 8, 4)[0] == 6);
00286     CHECK(processToLines(3, 8, 4)[1] == 7);
00287 }
```

Here is the call graph for this function:

6.13.2.18 TEST_CASE() [18/25]

```
TEST_CASE (
            "Number of lines assigned to a process:    p < n,
            ((n mod p) !=0)" ,
            "" [processToLinesCount] )
```

Unit test for method processToLinesCount with 7 lines and 4 processes (p < n), ((n mod p) != 0)

Definition at line 293 of file networking_unittest.cpp.

```
00293                                                 : (p < n), ((n mod p) != 0)", "[processToLinesCount]")
00294 {
00295     CHECK(processToLines(0, 7, 4).size() == 2);
00296     CHECK(processToLines(1, 7, 4).size() == 2);
00297     CHECK(processToLines(2, 7, 4).size() == 2);
00298     CHECK(processToLines(3, 7, 4).size() == 1);
00299     CHECK(processToLines(0, 7, 4)[0] == 0);
00300     CHECK(processToLines(0, 7, 4)[1] == 1);
00301     CHECK(processToLines(1, 7, 4)[0] == 2);
00302     CHECK(processToLines(1, 7, 4)[1] == 3);
00303     CHECK(processToLines(2, 7, 4)[0] == 4);
00304     CHECK(processToLines(2, 7, 4)[1] == 5);
00305     CHECK(processToLines(3, 7, 4)[0] == 6);
00306 }
```

Here is the call graph for this function:

6.13.2.19 TEST_CASE() [19/25]

```
TEST_CASE (
             "Number of lines assigned to a process:    p > n,
             ((n mod p)==0)" ,
             "" [processToLinesCount] )
```

Unit test for method processToLinesCount with 4 lines and 8 processes (p > n), ((n mod p) == 0)

Definition at line 312 of file networking_unittest.cpp.

```
00312                                                : (p > n), ((n mod p) == 0)", "[processToLinesCount]")
00313 {
00314     CHECK(processToLines(0, 4, 8).size() == 1);
00315     CHECK(processToLines(1, 4, 8).size() == 1);
00316     CHECK(processToLines(2, 4, 8).size() == 1);
00317     CHECK(processToLines(3, 4, 8).size() == 1);
00318     CHECK(processToLines(4, 4, 8).size() == 0);
00319     CHECK(processToLines(5, 4, 8).size() == 0);
00320     CHECK(processToLines(6, 4, 8).size() == 0);
00321     CHECK(processToLines(7, 4, 8).size() == 0);
00322     CHECK(processToLines(0, 4, 8)[0] == 0);
00323     CHECK(processToLines(1, 4, 8)[0] == 1);
00324     CHECK(processToLines(2, 4, 8)[0] == 2);
00325     CHECK(processToLines(3, 4, 8)[0] == 3);
00326 }
```

Here is the call graph for this function:

6.13.2.20 TEST_CASE() [20/25]

```
TEST_CASE (
             "Number of lines assigned to a process:    p > n,
             ((n mod p) !=0)" ,
             "" [processToLinesCount] )
```

Unit test for method processToLinesCount with 5 lines and 8 processes (p > n), ((n mod p) != 0)

Definition at line 332 of file networking_unittest.cpp.

```
00332                                                 : (p > n), ((n mod p) != 0)", "[processToLinesCount]")
00333 {
00334     CHECK(processToLines(0, 5, 8).size() == 1);
00335     CHECK(processToLines(1, 5, 8).size() == 1);
00336     CHECK(processToLines(2, 5, 8).size() == 1);
00337     CHECK(processToLines(3, 5, 8).size() == 1);
00338     CHECK(processToLines(4, 5, 8).size() == 1);
00339     CHECK(processToLines(5, 5, 8).size() == 0);
00340     CHECK(processToLines(6, 5, 8).size() == 0);
00341     CHECK(processToLines(7, 5, 8).size() == 0);
00342     CHECK(processToLines(0, 5, 8)[0] == 0);
00343     CHECK(processToLines(1, 5, 8)[0] == 1);
00344     CHECK(processToLines(2, 5, 8)[0] == 2);
00345     CHECK(processToLines(3, 5, 8)[0] == 3);
00346     CHECK(processToLines(4, 5, 8)[0] == 4);
00347 }
```

Here is the call graph for this function:

6.13.2.21 TEST_CASE() [21/25]

```
TEST_CASE (
             "Calculate didpls and scounts for MPI_Scatterv:    p == n,
             ((n mod p)==0)" ,
             "" [calculateDisplsScounts] )
```

Unit test for method calculateDisplsScounts with 4 lines and 4 processes (p == n), ((n mod p) == 0)

Definition at line 353 of file networking_unittest.cpp.

```
00353                                                     : (p == n), ((n mod p) == 0)",
      "[calculateDisplsScounts]")
00354 {
00355     int *displs, *scounts;
00356     int arraySize = 4;
00357     int processes = 4;
00358     int myRank = 0;
00359
00360     calculateDisplsScounts(displs, scounts, arraySize, processes, myRank);
00361
00362     CHECK(displs[0] == 0);
00363     CHECK(displs[1] == 4);
00364     CHECK(displs[2] == 8);
00365     CHECK(displs[3] == 12);
00366
00367     CHECK(scounts[0] == 4);
00368     CHECK(scounts[1] == 4);
00369     CHECK(scounts[2] == 4);
00370     CHECK(scounts[3] == 4);
00371
00372     delete[] displs;
00373     delete[] scounts;
00374 }
```

Here is the call graph for this function:

6.13.2.22 TEST_CASE() [22/25]

```
TEST_CASE (
            "Calculate didpls and scounts for MPI_Scatterv:    p < n,
            ((n mod p)==0)" ,
            "" [calculateDisplsScounts] )
```

Unit test for method calculateDisplsScounts with 8 lines and 4 processes (p < n), ((n mod p) == 0)

Definition at line 380 of file networking_unittest.cpp.

```
00380                                                    : (p < n), ((n mod p) == 0)",
      "[calculateDisplsScounts]")
00381 {
00382     int *displs, *scounts;
00383     int arraySize = 8;
00384     int processes = 4;
00385     int myRank = 0;
00386
00387     calculateDisplsScounts(displs, scounts, arraySize, processes, myRank);
00388
00389     CHECK(displs[0] == 0);
00390     CHECK(displs[1] == 16);
00391     CHECK(displs[2] == 32);
00392     CHECK(displs[3] == 48);
00393
00394     CHECK(scounts[0] == 16);
00395     CHECK(scounts[1] == 16);
00396     CHECK(scounts[2] == 16);
00397     CHECK(scounts[3] == 16);
00398
00399     delete[] displs;
00400     delete[] scounts;
00401 }
```

Here is the call graph for this function:

6.13.2.23 TEST_CASE() [23/25]

```
TEST_CASE (
            "Calculate didpls and scounts for MPI_Scatterv:    p < n,
            ((n mod p) !=0)" ,
            "" [calculateDisplsScounts] )
```

Unit test for method calculateDisplsScounts with 7 lines and 4 processes (p < n), ((n mod p) != 0)

Definition at line 407 of file networking_unittest.cpp.

```
00407                                                    : (p < n), ((n mod p) != 0)",
      "[calculateDisplsScounts]")
00408 {
00409     int *displs, *scounts;
00410     int arraySize = 7;
00411     int processes = 4;
00412     int myRank = 0;
```

```
00413
00414     calculateDisplsScounts(displs, scounts, arraySize, processes, myRank);
00415
00416     CHECK(displs[0] == 0);
00417     CHECK(displs[1] == 14);
00418     CHECK(displs[2] == 28);
00419     CHECK(displs[3] == 42);
00420
00421     CHECK(scounts[0] == 14);
00422     CHECK(scounts[1] == 14);
00423     CHECK(scounts[2] == 14);
00424     CHECK(scounts[3] == 7);
00425
00426     delete[] displs;
00427     delete[] scounts;
00428 }
```

Here is the call graph for this function:

6.13.2.24 TEST_CASE() [24/25]

```
TEST_CASE (
             "Calculate didpls and scounts for MPI_Scatterv:    p > n,
             ((n mod p)==0)" ,
             "" [calculateDisplsScounts] )
```

Unit test for method calculateDisplsScounts with 4 lines and 8 processes (p > n), ((n mod p) == 0)

Definition at line 434 of file networking_unittest.cpp.

```
00434                                                    : (p > n), ((n mod p) == 0)",
     "[calculateDisplsScounts]")
00435 {
00436     int *displs, *scounts;
00437     int arraySize = 4;
00438     int processes = 8;
00439     int myRank = 0;
00440
00441     calculateDisplsScounts(displs, scounts, arraySize, processes, myRank);
00442
00443     CHECK(displs[0] == 0);
00444     CHECK(displs[1] == 4);
00445     CHECK(displs[2] == 8);
00446     CHECK(displs[3] == 12);
00447
00448     CHECK(scounts[0] == 4);
00449     CHECK(scounts[1] == 4);
00450     CHECK(scounts[2] == 4);
00451     CHECK(scounts[3] == 4);
00452
00453     delete[] displs;
00454     delete[] scounts;
00455 }
```

Here is the call graph for this function:

6.13.2.25 TEST_CASE() [25/25]

```
TEST_CASE (
             "Calculate didpls and scounts for MPI_Scatterv:    p > n,
             ((n mod p) !=0)" ,
             "" [calculateDisplsScounts] )
```

Unit test for method calculateDisplsScounts with 5 lines and 8 processes (p > n), ((n mod p) != 0)

Definition at line 461 of file networking_unittest.cpp.

```
00461                                                    : (p > n), ((n mod p) != 0)",
       "[calculateDisplsScounts]")
00462 {
00463     int *displs, *scounts;
00464     int arraySize = 5;
00465     int processes = 8;
00466     int myRank = 0;
00467
00468     calculateDisplsScounts(displs, scounts, arraySize, processes, myRank);
00469
00470     CHECK(displs[0] == 0);
00471     CHECK(displs[1] == 5);
00472     CHECK(displs[2] == 10);
00473     CHECK(displs[3] == 15);
00474     CHECK(displs[4] == 20);
00475
00476     CHECK(scounts[0] == 5);
00477     CHECK(scounts[1] == 5);
00478     CHECK(scounts[2] == 5);
00479     CHECK(scounts[3] == 5);
00480     CHECK(scounts[4] == 5);
00481
00482     delete[] displs;
00483     delete[] scounts;
00484 }
```

Here is the call graph for this function:

6.14 networking_unittest.cpp

```
00001
00012 #define CATCH_CONFIG_MAIN
00013
00014 #include "../lib/catch.hpp"
00015
00016 #include "mpi.h"
00017
00018 #include "../src/networking.h"
00019
00024 TEST_CASE("[Internal] Line to Process Assignments: (p == n), ((n mod p) == 0)", "[ilineToProcess]")
00025 {
00026     CHECK(_lineToProcess(0, 4, 4) == 0);
00027     CHECK(_lineToProcess(1, 4, 4) == 1);
00028     CHECK(_lineToProcess(2, 4, 4) == 2);
00029     CHECK(_lineToProcess(3, 4, 4) == 3);
00030 }
00031
00036 TEST_CASE("[Internal] Line to Process Assignments: (p < n), ((n mod p) == 0)", "[ilineToProcess]")
00037 {
00038     CHECK(_lineToProcess(0, 8, 4) == 0);
00039     CHECK(_lineToProcess(1, 8, 4) == 1);
00040     CHECK(_lineToProcess(2, 8, 4) == 2);
00041     CHECK(_lineToProcess(3, 8, 4) == 3);
00042     CHECK(_lineToProcess(4, 8, 4) == 0);
00043     CHECK(_lineToProcess(5, 8, 4) == 1);
00044     CHECK(_lineToProcess(6, 8, 4) == 2);
```

```
00045     CHECK(_lineToProcess(7, 8, 4) == 3);
00046 }
00047
00052 TEST_CASE("[Internal] Line to Process Assignments: (p < n), ((n mod p) != 0)", "[ilineToProcess]")
00053 {
00054     CHECK(_lineToProcess(0, 7, 4) == 0);
00055     CHECK(_lineToProcess(1, 7, 4) == 1);
00056     CHECK(_lineToProcess(2, 7, 4) == 2);
00057     CHECK(_lineToProcess(3, 7, 4) == 3);
00058     CHECK(_lineToProcess(4, 7, 4) == 0);
00059     CHECK(_lineToProcess(5, 7, 4) == 1);
00060     CHECK(_lineToProcess(6, 7, 4) == 2);
00061 }
00062
00067 TEST_CASE("[Internal] Line to Process Assignments: (p > n), ((n mod p) == 0)", "[ilineToProcess]")
00068 {
00069     CHECK(_lineToProcess(0, 4, 8) == 0);
00070     CHECK(_lineToProcess(1, 4, 8) == 1);
00071     CHECK(_lineToProcess(2, 4, 8) == 2);
00072     CHECK(_lineToProcess(3, 4, 8) == 3);
00073 }
00074
00079 TEST_CASE("[Internal] Line to Process Assignments: (p > n), ((n mod p) != 0)", "[ilineToProcess]")
00080 {
00081     CHECK(_lineToProcess(0, 5, 8) == 0);
00082     CHECK(_lineToProcess(1, 5, 8) == 1);
00083     CHECK(_lineToProcess(2, 5, 8) == 2);
00084     CHECK(_lineToProcess(3, 5, 8) == 3);
00085     CHECK(_lineToProcess(4, 5, 8) == 4);
00086 }
00087
00092 TEST_CASE("Line to Process Assignments: (p == n), ((n mod p) == 0)", "[lineToProcess]")
00093 {
00094     CHECK(lineToProcess(0, 4, 4) == 0);
00095     CHECK(lineToProcess(1, 4, 4) == 1);
00096     CHECK(lineToProcess(2, 4, 4) == 2);
00097     CHECK(lineToProcess(3, 4, 4) == 3);
00098 }
00099
00104 TEST_CASE("Line to Process Assignments: (p < n), ((n mod p) == 0)", "[lineToProcess]")
00105 {
00106     CHECK(lineToProcess(0, 8, 4) == 0);
00107     CHECK(lineToProcess(1, 8, 4) == 0);
00108     CHECK(lineToProcess(2, 8, 4) == 1);
00109     CHECK(lineToProcess(3, 8, 4) == 1);
00110     CHECK(lineToProcess(4, 8, 4) == 2);
00111     CHECK(lineToProcess(5, 8, 4) == 2);
00112     CHECK(lineToProcess(6, 8, 4) == 3);
00113     CHECK(lineToProcess(7, 8, 4) == 3);
00114 }
00115
00120 TEST_CASE("Line to Process Assignments: (p < n), ((n mod p) != 0)", "[lineToProcess]")
00121 {
00122     CHECK(lineToProcess(0, 7, 4) == 0);
00123     CHECK(lineToProcess(1, 7, 4) == 0);
00124     CHECK(lineToProcess(2, 7, 4) == 1);
00125     CHECK(lineToProcess(3, 7, 4) == 1);
00126     CHECK(lineToProcess(4, 7, 4) == 2);
00127     CHECK(lineToProcess(5, 7, 4) == 2);
00128     CHECK(lineToProcess(6, 7, 4) == 3);
00129 }
00130
00135 TEST_CASE("Line to Process Assignments: (p > n), ((n mod p) == 0)", "[lineToProcess]")
00136 {
00137     CHECK(lineToProcess(0, 4, 8) == 0);
00138     CHECK(lineToProcess(1, 4, 8) == 1);
00139     CHECK(lineToProcess(2, 4, 8) == 2);
00140     CHECK(lineToProcess(3, 4, 8) == 3);
00141 }
00142
00147 TEST_CASE("Line to Process Assignments: (p > n), ((n mod p) != 0)", "[lineToProcess]")
00148 {
00149     CHECK(lineToProcess(0, 5, 8) == 0);
00150     CHECK(lineToProcess(1, 5, 8) == 1);
00151     CHECK(lineToProcess(2, 5, 8) == 2);
00152     CHECK(lineToProcess(3, 5, 8) == 3);
00153     CHECK(lineToProcess(4, 5, 8) == 4);
00154 }
00155
00160 TEST_CASE("Lines assigned to a process: (p == n), ((n mod p) == 0)", "[processToLines]")
00161 {
00162     CHECK(processToLines(0, 4, 4).size() == 1);
00163     CHECK(processToLines(1, 4, 4).size() == 1);
00164     CHECK(processToLines(2, 4, 4).size() == 1);
00165     CHECK(processToLines(3, 4, 4).size() == 1);
00166
00167     CHECK(processToLines(0, 4, 4)[0] == 0);
```

```
00168     CHECK(processToLines(1, 4, 4)[0] == 1);
00169     CHECK(processToLines(2, 4, 4)[0] == 2);
00170     CHECK(processToLines(3, 4, 4)[0] == 3);
00171 }
00172
00177 TEST_CASE("Lines assigned to a process: (p < n), ((n mod p) == 0)", "[processToLines]")
00178 {
00179     CHECK(processToLines(0, 8, 4).size() == 2);
00180     CHECK(processToLines(1, 8, 4).size() == 2);
00181     CHECK(processToLines(2, 8, 4).size() == 2);
00182     CHECK(processToLines(3, 8, 4).size() == 2);
00183     CHECK(processToLines(0, 8, 4)[0] == 0);
00184     CHECK(processToLines(0, 8, 4)[1] == 1);
00185     CHECK(processToLines(1, 8, 4)[0] == 2);
00186     CHECK(processToLines(1, 8, 4)[1] == 3);
00187     CHECK(processToLines(2, 8, 4)[0] == 4);
00188     CHECK(processToLines(2, 8, 4)[1] == 5);
00189     CHECK(processToLines(3, 8, 4)[0] == 6);
00190     CHECK(processToLines(3, 8, 4)[1] == 7);
00191 }
00192
00197 TEST_CASE("Lines assigned to a process: (p < n), ((n mod p) != 0)", "[processToLines]")
00198 {
00199     CHECK(processToLines(0, 7, 4).size() == 2);
00200     CHECK(processToLines(1, 7, 4).size() == 2);
00201     CHECK(processToLines(2, 7, 4).size() == 2);
00202     CHECK(processToLines(3, 7, 4).size() == 1);
00203     CHECK(processToLines(0, 7, 4)[0] == 0);
00204     CHECK(processToLines(0, 7, 4)[1] == 1);
00205     CHECK(processToLines(1, 7, 4)[0] == 2);
00206     CHECK(processToLines(1, 7, 4)[1] == 3);
00207     CHECK(processToLines(2, 7, 4)[0] == 4);
00208     CHECK(processToLines(2, 7, 4)[1] == 5);
00209     CHECK(processToLines(3, 7, 4)[0] == 6);
00210 }
00211
00216 TEST_CASE("Lines assigned to a process: (p > n), ((n mod p) == 0)", "[processToLines]")
00217 {
00218     CHECK(processToLines(0, 4, 8).size() == 1);
00219     CHECK(processToLines(1, 4, 8).size() == 1);
00220     CHECK(processToLines(2, 4, 8).size() == 1);
00221     CHECK(processToLines(3, 4, 8).size() == 1);
00222     CHECK(processToLines(4, 4, 8).size() == 0);
00223     CHECK(processToLines(5, 4, 8).size() == 0);
00224     CHECK(processToLines(6, 4, 8).size() == 0);
00225     CHECK(processToLines(7, 4, 8).size() == 0);
00226     CHECK(processToLines(0, 4, 8)[0] == 0);
00227     CHECK(processToLines(1, 4, 8)[0] == 1);
00228     CHECK(processToLines(2, 4, 8)[0] == 2);
00229     CHECK(processToLines(3, 4, 8)[0] == 3);
00230 }
00231
00236 TEST_CASE("Lines assigned to a process: (p > n), ((n mod p) != 0)", "[processToLines]")
00237 {
00238     CHECK(processToLines(0, 5, 8).size() == 1);
00239     CHECK(processToLines(1, 5, 8).size() == 1);
00240     CHECK(processToLines(2, 5, 8).size() == 1);
00241     CHECK(processToLines(3, 5, 8).size() == 1);
00242     CHECK(processToLines(4, 5, 8).size() == 1);
00243     CHECK(processToLines(5, 5, 8).size() == 0);
00244     CHECK(processToLines(6, 5, 8).size() == 0);
00245     CHECK(processToLines(7, 5, 8).size() == 0);
00246     CHECK(processToLines(0, 5, 8)[0] == 0);
00247     CHECK(processToLines(1, 5, 8)[0] == 1);
00248     CHECK(processToLines(2, 5, 8)[0] == 2);
00249     CHECK(processToLines(3, 5, 8)[0] == 3);
00250     CHECK(processToLines(4, 5, 8)[0] == 4);
00251 }
00252
00257 TEST_CASE("Number of lines assigned to a process: (p == n), ((n mod p) == 0)",
      "[processToLinesCount]")
00258 {
00259     CHECK(processToLines(0, 4, 4).size() == 1);
00260     CHECK(processToLines(1, 4, 4).size() == 1);
00261     CHECK(processToLines(2, 4, 4).size() == 1);
00262     CHECK(processToLines(3, 4, 4).size() == 1);
00263     CHECK(processToLines(0, 4, 4)[0] == 0);
00264     CHECK(processToLines(1, 4, 4)[0] == 1);
00265     CHECK(processToLines(2, 4, 4)[0] == 2);
00266     CHECK(processToLines(3, 4, 4)[0] == 3);
00267 }
00268
00273 TEST_CASE("Number of lines assigned to a process: (p < n), ((n mod p) == 0)", "[processToLinesCount]")
00274 {
00275     CHECK(processToLines(0, 8, 4).size() == 2);
00276     CHECK(processToLines(1, 8, 4).size() == 2);
00277     CHECK(processToLines(2, 8, 4).size() == 2);
```

```
00278     CHECK(processToLines(3, 8, 4).size() == 2);
00279     CHECK(processToLines(0, 8, 4)[0] == 0);
00280     CHECK(processToLines(0, 8, 4)[1] == 1);
00281     CHECK(processToLines(1, 8, 4)[0] == 2);
00282     CHECK(processToLines(1, 8, 4)[1] == 3);
00283     CHECK(processToLines(2, 8, 4)[0] == 4);
00284     CHECK(processToLines(2, 8, 4)[1] == 5);
00285     CHECK(processToLines(3, 8, 4)[0] == 6);
00286     CHECK(processToLines(3, 8, 4)[1] == 7);
00287 }
00288
00293 TEST_CASE("Number of lines assigned to a process: (p < n), ((n mod p) != 0)", "[processToLinesCount]")
00294 {
00295     CHECK(processToLines(0, 7, 4).size() == 2);
00296     CHECK(processToLines(1, 7, 4).size() == 2);
00297     CHECK(processToLines(2, 7, 4).size() == 2);
00298     CHECK(processToLines(3, 7, 4).size() == 1);
00299     CHECK(processToLines(0, 7, 4)[0] == 0);
00300     CHECK(processToLines(0, 7, 4)[1] == 1);
00301     CHECK(processToLines(1, 7, 4)[0] == 2);
00302     CHECK(processToLines(1, 7, 4)[1] == 3);
00303     CHECK(processToLines(2, 7, 4)[0] == 4);
00304     CHECK(processToLines(2, 7, 4)[1] == 5);
00305     CHECK(processToLines(3, 7, 4)[0] == 6);
00306 }
00307
00312 TEST_CASE("Number of lines assigned to a process: (p > n), ((n mod p) == 0)", "[processToLinesCount]")
00313 {
00314     CHECK(processToLines(0, 4, 8).size() == 1);
00315     CHECK(processToLines(1, 4, 8).size() == 1);
00316     CHECK(processToLines(2, 4, 8).size() == 1);
00317     CHECK(processToLines(3, 4, 8).size() == 1);
00318     CHECK(processToLines(4, 4, 8).size() == 0);
00319     CHECK(processToLines(5, 4, 8).size() == 0);
00320     CHECK(processToLines(6, 4, 8).size() == 0);
00321     CHECK(processToLines(7, 4, 8).size() == 0);
00322     CHECK(processToLines(0, 4, 8)[0] == 0);
00323     CHECK(processToLines(1, 4, 8)[0] == 1);
00324     CHECK(processToLines(2, 4, 8)[0] == 2);
00325     CHECK(processToLines(3, 4, 8)[0] == 3);
00326 }
00327
00332 TEST_CASE("Number of lines assigned to a process: (p > n), ((n mod p) != 0)", "[processToLinesCount]")
00333 {
00334     CHECK(processToLines(0, 5, 8).size() == 1);
00335     CHECK(processToLines(1, 5, 8).size() == 1);
00336     CHECK(processToLines(2, 5, 8).size() == 1);
00337     CHECK(processToLines(3, 5, 8).size() == 1);
00338     CHECK(processToLines(4, 5, 8).size() == 1);
00339     CHECK(processToLines(5, 5, 8).size() == 0);
00340     CHECK(processToLines(6, 5, 8).size() == 0);
00341     CHECK(processToLines(7, 5, 8).size() == 0);
00342     CHECK(processToLines(0, 5, 8)[0] == 0);
00343     CHECK(processToLines(1, 5, 8)[0] == 1);
00344     CHECK(processToLines(2, 5, 8)[0] == 2);
00345     CHECK(processToLines(3, 5, 8)[0] == 3);
00346     CHECK(processToLines(4, 5, 8)[0] == 4);
00347 }
00348
00353 TEST_CASE("Calculate didpls and scounts for MPI_Scatterv: (p == n), ((n mod p) == 0)",
      "[calculateDisplsScounts]")
00354 {
00355     int *displs, *scounts;
00356     int arraySize = 4;
00357     int processes = 4;
00358     int myRank = 0;
00359
00360     calculateDisplsScounts(displs, scounts, arraySize, processes, myRank);
00361
00362     CHECK(displs[0] == 0);
00363     CHECK(displs[1] == 4);
00364     CHECK(displs[2] == 8);
00365     CHECK(displs[3] == 12);
00366
00367     CHECK(scounts[0] == 4);
00368     CHECK(scounts[1] == 4);
00369     CHECK(scounts[2] == 4);
00370     CHECK(scounts[3] == 4);
00371
00372     delete[] displs;
00373     delete[] scounts;
00374 }
00375
00380 TEST_CASE("Calculate didpls and scounts for MPI_Scatterv: (p < n), ((n mod p) == 0)",
      "[calculateDisplsScounts]")
00381 {
00382     int *displs, *scounts;
```

```
00383     int arraySize = 8;
00384     int processes = 4;
00385     int myRank = 0;
00386
00387     calculateDisplsScounts(displs, scounts, arraySize, processes, myRank);
00388
00389     CHECK(displs[0] == 0);
00390     CHECK(displs[1] == 16);
00391     CHECK(displs[2] == 32);
00392     CHECK(displs[3] == 48);
00393
00394     CHECK(scounts[0] == 16);
00395     CHECK(scounts[1] == 16);
00396     CHECK(scounts[2] == 16);
00397     CHECK(scounts[3] == 16);
00398
00399     delete[] displs;
00400     delete[] scounts;
00401 }
00402
00407 TEST_CASE("Calculate didpls and scounts for MPI_Scatterv: (p < n), ((n mod p) != 0)",
      "[calculateDisplsScounts]")
00408 {
00409     int *displs, *scounts;
00410     int arraySize = 7;
00411     int processes = 4;
00412     int myRank = 0;
00413
00414     calculateDisplsScounts(displs, scounts, arraySize, processes, myRank);
00415
00416     CHECK(displs[0] == 0);
00417     CHECK(displs[1] == 14);
00418     CHECK(displs[2] == 28);
00419     CHECK(displs[3] == 42);
00420
00421     CHECK(scounts[0] == 14);
00422     CHECK(scounts[1] == 14);
00423     CHECK(scounts[2] == 14);
00424     CHECK(scounts[3] == 7);
00425
00426     delete[] displs;
00427     delete[] scounts;
00428 }
00429
00434 TEST_CASE("Calculate didpls and scounts for MPI_Scatterv: (p > n), ((n mod p) == 0)",
      "[calculateDisplsScounts]")
00435 {
00436     int *displs, *scounts;
00437     int arraySize = 4;
00438     int processes = 8;
00439     int myRank = 0;
00440
00441     calculateDisplsScounts(displs, scounts, arraySize, processes, myRank);
00442
00443     CHECK(displs[0] == 0);
00444     CHECK(displs[1] == 4);
00445     CHECK(displs[2] == 8);
00446     CHECK(displs[3] == 12);
00447
00448     CHECK(scounts[0] == 4);
00449     CHECK(scounts[1] == 4);
00450     CHECK(scounts[2] == 4);
00451     CHECK(scounts[3] == 4);
00452
00453     delete[] displs;
00454     delete[] scounts;
00455 }
00456
00461 TEST_CASE("Calculate didpls and scounts for MPI_Scatterv: (p > n), ((n mod p) != 0)",
      "[calculateDisplsScounts]")
00462 {
00463     int *displs, *scounts;
00464     int arraySize = 5;
00465     int processes = 8;
00466     int myRank = 0;
00467
00468     calculateDisplsScounts(displs, scounts, arraySize, processes, myRank);
00469
00470     CHECK(displs[0] == 0);
00471     CHECK(displs[1] == 5);
00472     CHECK(displs[2] == 10);
00473     CHECK(displs[3] == 15);
00474     CHECK(displs[4] == 20);
00475
00476     CHECK(scounts[0] == 5);
00477     CHECK(scounts[1] == 5);
00478     CHECK(scounts[2] == 5);
```

```
00479     CHECK(scounts[3] == 5);
00480     CHECK(scounts[4] == 5);
00481
00482     delete[] displs;
00483     delete[] scounts;
00484 }
00485
```

Index