CPPExercise

Generated by Doxygen 1.8.8

Sun May 1 2016 19:10:37

Contents

Chapter 1

C++ Programming Exercise

Instructions

Produce a small working application for review by your prospective employer, the details of the application are provided below.

The aim for the test is to understand how you structure your solution, and how well you manage to turn that solution into a working application.

You need to:

- Produce working Object Orientated, unit-tested code.
- Provide instructions on running and building your application.
- Deliver the working application to your prospective employer.

You are expected to complete the test alone without immediate help, however should you have any questions or need more clarity then contact via Github either by raising an issue or messaging directly.

You are not expected to know all the technologies used in this exercise, however you should be able to find information on the internet.

There is no given time limit, however the test is designed to be short. Please read the entire brief and then contact the person who gave you this assignment if you feel you are unable to complete the task in the allotted time frame.

The Assignment

Before you begin to write your solution, you should first fork this repository. Provide a link to your forked repository to the person who gave you this exercise. You should (as with any project) commit and push regularly to your forked repository.

The solution should be written in C++.

A boilerplate project has been provided as part of the repository. Candidates are encouraged to use the provided framework as the basis of their project.

The boilerplate project makes use of and uses the following dependencies:

- CMake
- googletest
- git

A `Vagrantfile` has been provided, it can optionally be used to build the project using Centos 6.

If any of the above software is unknown to you, then you are expected to research the subjects yourself, factor this in to the estimate you give to complete this task.

The project has been designed with Cross-platform support in mind. Ensure that your software builds and runs on at minimum `Centos 6`, and preferably `Windows 2008 r2 or greater` and/or `Mac OS 10`.

Efforts should be made maintain support for the following platforms:

- Windows 2008 r2 or greater
- Centos 6 or other linux
- Mac OS 10

We do not expect that you may have access to all of the OS platforms listed above. Should you be unable to test on all of the above platforms make a note of this fact here: MacOS

The goal of this exercise is to produce a reusable library which provides basic calculator functions.

The calculator library should have methods for the following functionality:

- Add a value
- Subtract a value
- Multiply by a value
- Divide by a value
- A method to return the total
- An undo method

The library should have written tests, but also a simple console application which demonstrates the library should also be provided.

The application should:

- Accept operations from stdin
- Print to stdout the running total
- a command to display the command history
- a command to undo
- a command to clear/restart

Documentation on both the library and the console application should be included.

Building the boilerplate project

Linux / MacOS

The following commands will build on linux:

“‘ mkdir build cd build cmake ../ make make test “‘

This project requires googletest, googletest can be installed system-wide or locally, if installed locally set the `GT↩EST_ROOT` variable to the install location of googletest.

Note that the supplied Vagrantfile will build a linux VM and provision it for building the project.

Windows

There are various ways to build on windows. This project has been tested using MS Visual Studio Express 2010. Using other versions of Visual Studio and other Generators should work fine as well.

Download and install CMake make sure it is in your PATH.

Download and unpack Google Test to `c:\googletest`

Build googletest from the `Visual Studio Command Prompt (2010):`

“‘ cd c: cmake ./ -G "NMake Makefiles" nmake “‘

The CMake variable `GTEST_ROOT` can be configured should googletest be installed in a different location.

To build this project still from the `Visual Studio Command Prompt (2010):`

“‘ cd c:<location of this repo> mkdir build cd build cmake ../ -G "NMake Makefiles" nmake nmake test “‘

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 Calculator Class Reference

This is the class that performs and holds the performed operations.

`#include <Calculator.h>`

Classes

- class DivisionByZeroException

  This class represents the exception thrown by this class when a division by zero is attempted.

Public Member Functions

- Calculator ()

  The default constructor.
- Calculator (const double &initialValue)

  The constructor that allows to set a different initial value.
- double getCurrentTotal ()

  Returns the current running total.
- double addValue (const double &value)

  Issues an addition operation with the specified operand.
- double subtractValue (const double &value)

  Issues a subtraction operation with the specified operand.
- double multiplyBy (const double &value)

  Issues a multiplication operation with the specified operand.
- double divideBy (const double &value)

  Issues an addition operation with the specified operand.
- double getPreviousTotal ()

  Returns the total prior to performing the last operation. It calls the same method as undo, but does not update the currentTotal, nor does it remove the operations from the operationsList vector.
- double undo ()

  Undoes the last performed operation. This method calls undo on the last performed operation, which is held at the back of the operationList vector. The currentTotal is updated with the value returned. Then the last operation is removed from the operationList.
- std::string getOperationsString ()

  Returns the command history in string form. This method calls the Operation::toString() methods of all the consecutive Operation objects held in the vector. The results are being appended to the end of the string object holding the calculators initial value.

- void clear ()

  Resets the calculator. This method clears the operationList vector, resets the currentTotal to 0, and updates the initialStringStream with the new total.

5.1.1 Detailed Description

This is the class that performs and holds the performed operations.

Definition at line 16 of file Calculator.h.

5.1.2 Member Function Documentation

5.1.2.1 double Calculator::addValue (const double & *value*)

Issues an addition operation with the specified operand.

This method creates a new Operation object with OPERATION_TYPE::SUM set as its operation type, and the operand passed to it. The operation is then added to the operationList vector and the Operation::perform() method is called with the currentTotal passed to it. The result is saved in the currentTotal member.

Parameters

in	*value*	the operand value

Returns

the calculated result of the operation

See also

Operation

Definition at line 27 of file Calculator.cpp.

5.1.2.2 double Calculator::divideBy (const double & *value*)

Issues an addition operation with the specified operand.

This method acts similarly to addValue, but creates a new Operation object with Operation::OPERATION_TYPE↩ ::DIVISION set as its operation type. If the operand is equal to zero, a DivisionByZeroException is thrown.

Parameters

in	*value*	the operand value

Returns

the calculated result of the operation

See also

Operation, DivisionByZeroException

Definition at line 51 of file Calculator.cpp.

5.1.2.3 double Calculator::getCurrentTotal ( )

Returns the current running total.

Returns

the current total

Definition at line 22 of file Calculator.cpp.

5.1.2.4 std::string Calculator::getOperationsString ( )

Returns the command history in string form. This method calls the Operation::toString() methods of all the consecutive Operation objects held in the vector. The results are being appended to the end of the string object holding the calculators initial value.

Returns

a string holding the command history

See also

Operation

Definition at line 78 of file Calculator.cpp.

5.1.2.5 double Calculator::getPreviousTotal ( )

Returns the total prior to performing the last operation. It calls the same method as undo, but does not update the currentTotal, nor does it remove the operations from the operationsList vector.

Returns

the total prior to performing the last operation

Definition at line 62 of file Calculator.cpp.

5.1.2.6 double Calculator::multiplyBy (const double & *value*)

Issues a multiplication operation with the specified operand.

This method acts similarly to addValue, but creates a new Operation object with Operation::OPERATION_TYPE↩ ::MULTIPLICATION set as its operation type.

Parameters

in	*value*	the operand value

Returns

the calculated result of the operation

See also

Operation

Definition at line 43 of file Calculator.cpp.

5.1.2.7 double Calculator::subtractValue (const double & *value*)

Issues a subtraction operation with the specified operand.

This method acts similarly to addValue, but creates a new Operation object with Operation::OPERATION_TYPE↩ ::SUBTRACTION set as its operation type.

Parameters

in	*value*	the operand value

Returns

the calculated result of the operation

See also

Operation

Definition at line 35 of file Calculator.cpp.

5.1.2.8 double Calculator::undo ( )

Undoes the last performed operation. This method calls undo on the last performed operation, which is held at the back of the operationList vector. The currentTotal is updated with the value returned. Then the last operation is removed from the operationList.

Returns

the total after undoing the operation

Definition at line 69 of file Calculator.cpp.

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

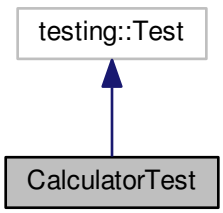
- headers/Calculator.h
- sources/Calculator.cpp

5.2 CalculatorTest Class Reference

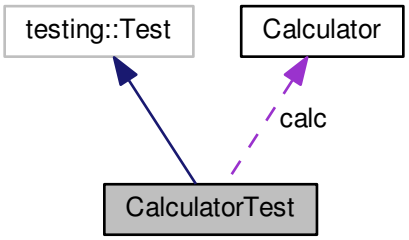
This class is a test fixture, that sets up the necessary data before performing tests of the Calculator class.

`#include <calculatorTest.h>`

Inheritance diagram for CalculatorTest:



Collaboration diagram for CalculatorTest:



Protected Member Functions

- virtual void SetUp ()

  Performs the setup before running each and every of the associated tests.
- virtual void TearDown ()

  Performs the removal of resources after any of the associated tests finish.

Static Protected Member Functions

- static void SetUpTestCase ()

  Performs the setup before running any of the associated tests.
- static void TearDownTestCase ()

  Performs the removal of resources after every associated test has been finished.
- static void createNewCalculator (const double &initialValue)

  Creates a new calculator object and initializes it with the given value.

Static Protected Attributes

- static Calculator * calc

5.2.1 Detailed Description

This class is a test fixture, that sets up the necessary data before performing tests of the Calculator class.

Definition at line 11 of file calculatorTest.h.

5.2.2 Member Function Documentation

5.2.2.1 void CalculatorTest::createNewCalculator (const double & *initialValue*) [static], [protected]

Creates a new calculator object and initializes it with the given value.

Parameters

in	*initialValue*	the initial calculator value

Definition at line 29 of file calculatorTest.cpp.

5.2.3 Member Data Documentation

5.2.3.1 Calculator * CalculatorTest::calc [static], [protected]

A calculator object used in the tests.

Definition at line 36 of file calculatorTest.h.

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

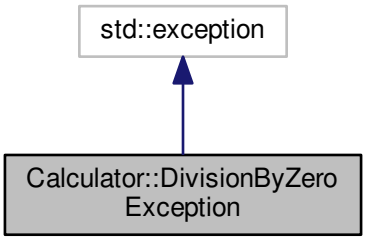
- tests/calculatorTest.h
- tests/calculatorTest.cpp

5.3 Calculator::DivisionByZeroException Class Reference

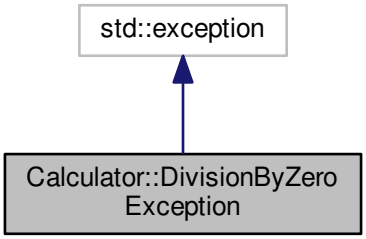
This class represents the exception thrown by this class when a division by zero is attempted.

`#include <Calculator.h>`

Inheritance diagram for Calculator::DivisionByZeroException:



Collaboration diagram for Calculator::DivisionByZeroException:



Public Member Functions

- virtual const char * what () const throw ()

  Returns the exception reson.

5.3.1 Detailed Description

This class represents the exception thrown by this class when a division by zero is attempted.

Definition at line 29 of file Calculator.h.

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

- headers/Calculator.h

5.4 Operation Class Reference

A class that handles the operations performed by the calculator.

`#include <Operation.h>`

Public Types

- enum OPERATION_TYPE {
  SUM, SUBTRACTION, MULTIPLICATION, DIVISION,
  ERROR }

Public Member Functions

- Operation (const OPERATION_TYPE newOperationType, const double &newOperandValue)

  The constructor.
- double perform (const double &currentValue)

  Performs the operation on a value.
- double undo (const double &currentValue)

  Returns the initial value from before performing the operation.
- double performReverse (const double &currentValue)

Reverses the operation on the given argument.

- std::string toString ()

  Returns the operation symbol and value.

5.4.1 Detailed Description

A class that handles the operations performed by the calculator.

Definition at line 14 of file Operation.h.

5.4.2 Member Enumeration Documentation

5.4.2.1 enum Operation::OPERATION_TYPE

An enum holding the defined operation types.

Enumerator

SUM Instances that hold this value as the
See also
{operationType} represent the addition operation.

SUBTRACTION Instances that hold this value as the
See also
{operationType} represent the subtraction operation.

MULTIPLICATION Instances that hold this value as the
See also
{operationType} represent the multiplication operation.

DIVISION Instances that hold this value as the
See also
{operationType} represent the division operation.

ERROR Instances that hold this value as the
See also
{operationType} represent an errorneous operation.

Definition at line 18 of file Operation.h.

5.4.3 Constructor & Destructor Documentation

5.4.3.1 Operation::Operation (const OPERATION_TYPE *newOperationType*, const double & *newOperandValue*)

The constructor.

Constructor that requires specifying the member initial values.

Parameters

in	*newOperationType*	the type of the operation performed, should be one of the values from OPERATION_TYPE

in	*newOperand↩ Value*	the operand value

Definition at line 6 of file Operation.cpp.

5.4.4 Member Function Documentation

5.4.4.1 double Operation::perform (const double & *currentValue*)

Performs the operation on a value.

This method performes the operation set for this instance with the given operand. It also saves the given value in the valueBeforeOperation member. The return value is the calculated result. In case of dividing by zero, no operation takes place, and the return value is equal to the given argument.

Parameters

in	*currentValue*	the value on which the operation is to be performed

Returns

the calculated result

Definition at line 16 of file Operation.cpp.

5.4.4.2 double Operation::performReverse (const double & *currentValue*)

Reverses the operation on the given argument.

This method uses its operand to perform the opposite operation to the one set as this instances' operation type. This was initialy used as the undo method, but using it meant that any approximation errors introduced by an operation weren't undone.

Parameters

currentValue	the value on which the opposite operation is to be performed

Returns

the calculated result

Definition at line 41 of file Operation.cpp.

5.4.4.3 std::string Operation::toString ( )

Returns the operation symbol and value.

Returns

the operation symbol and value as a std::string object

Definition at line 60 of file Operation.cpp.

5.4.4.4 double Operation::undo (const double & *currentValue*)

Returns the initial value from before performing the operation.

This method returns the on which the operation was last performed on.

Parameters

currentValue	the value before undoing the operation; not used

Returns

the argument given when the operation was last performed

Definition at line 36 of file Operation.cpp.

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

- headers/Operation.h
- sources/Operation.cpp

Chapter 6

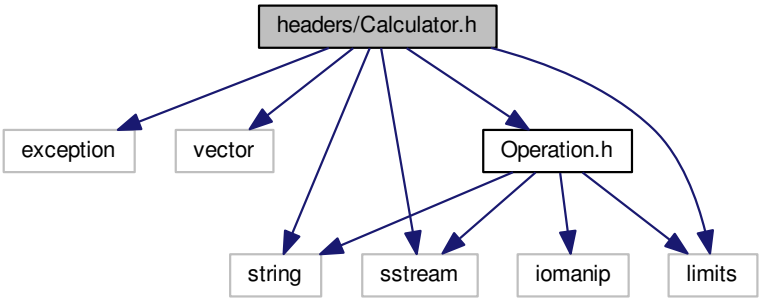
File Documentation

6.1 headers/Calculator.h File Reference

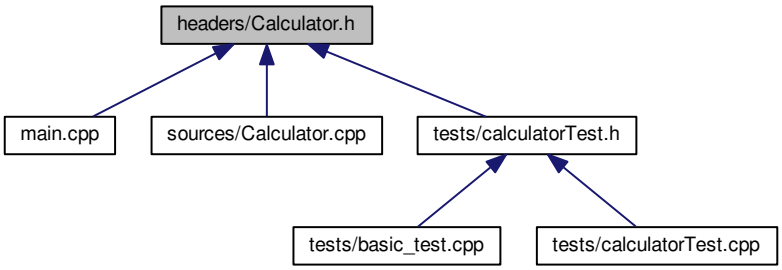
A file containing the Calculator class.

```
#include <exception>
#include <vector>
#include <string>
#include <sstream>
#include <limits>
#include <Operation.h>
```

Include dependency graph for Calculator.h:



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



Classes

- class Calculator

  This is the class that performs and holds the performed operations.
- class Calculator::DivisionByZeroException

  This class represents the exception thrown by this class when a division by zero is attempted.

6.1.1 Detailed Description

A file containing the Calculator class.

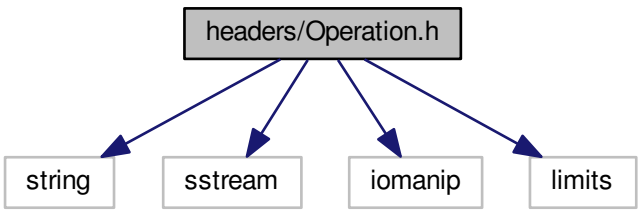
Definition in file Calculator.h.

6.2 headers/Operation.h File Reference

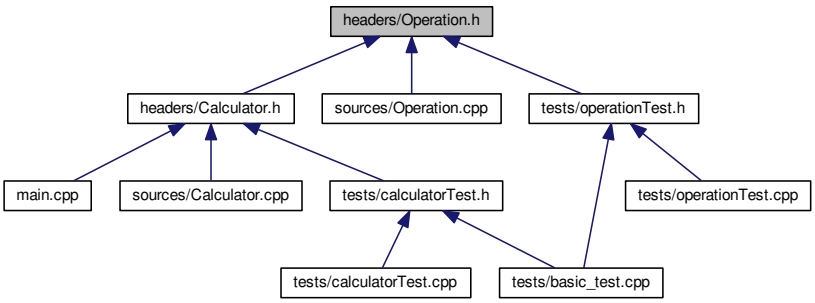
A file containing the Operation class.

```
#include <string>
#include <sstream>
#include <iomanip>
#include <limits>
```

Include dependency graph for Operation.h:



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



Classes

- class Operation

  A class that handles the operations performed by the calculator.

6.2.1 Detailed Description

A file containing the Operation class.

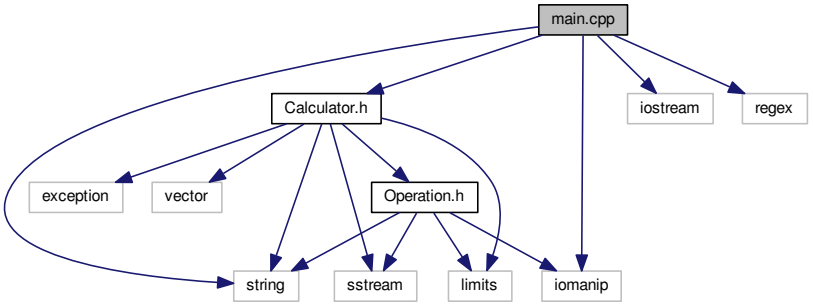
Definition in file Operation.h.

6.3 main.cpp File Reference

This file holds the application demonstrating the library.

```
#include <Calculator.h>
#include <iostream>
#include <string>
#include <iomanip>
#include <regex>
```

Include dependency graph for main.cpp:



Functions

- int getValueFromCommandString (const int &offset, string &input, int &numberOfCharsRead, double &value)

  Extracts a single numerical value from the given input string.
- int **main** (int argc, char ∗∗argv)

6.3.1 Detailed Description

This file holds the application demonstrating the library.

This application takes the user input command/chain of commands, and performs them sequentially. The numeric arguments can be written in regular or scientific format. The commands should have the following format:

- +FLOATING_POINT_VALUE - performs an addition operation with the given FLOATING_POINT_VALUE
- -FLOATING_POINT_VALUE - performs an subtraction operation with the given FLOATING_POINT_VALUE
- ∗FLOATING_POINT_VALUE - performs an multiplication operation with the given FLOATING_POINT_VA↩ LUE
- xFLOATING_POINT_VALUE - performs an multiplication operation with the given FLOATING_POINT_VA↩ LUE
- /FLOATING_POINT_VALUE - performs an division operation with the given FLOATING_POINT_VALUE
- h - displays the command history
- c - resets the calculator
- u - undoes the last operation
- q - exits the application

If a number is provided without any of the listed arithmetic operators, the calculator resets, and the given value is added to the initial calculator value.

The commands can be chained as follows:

+12.4∗2/4+8c12/4∗3

This will perform the operation in the order in which they were written in (it disregards the mathematical order of operations). In this example the result will be 9, because upon reaching the letter 'c', the calculator will reset.

Definition in file main.cpp.

6.3.2 Function Documentation

6.3.2.1 int getValueFromCommandString (const int & *offset*, string & *input*, int & *numberOfCharsRead*, double & *value*)

Extracts a single numerical value from the given input string.

This function processes the input, starting from the character number specified by offset. The found value is written to the value parameter.

Parameters

in	*offset*	the character number that the analysis should begin on
in	*input*	the parsed string

out	*numberOf↩ CharsRead*	the number of read characters
out	*value*	the numerical value found

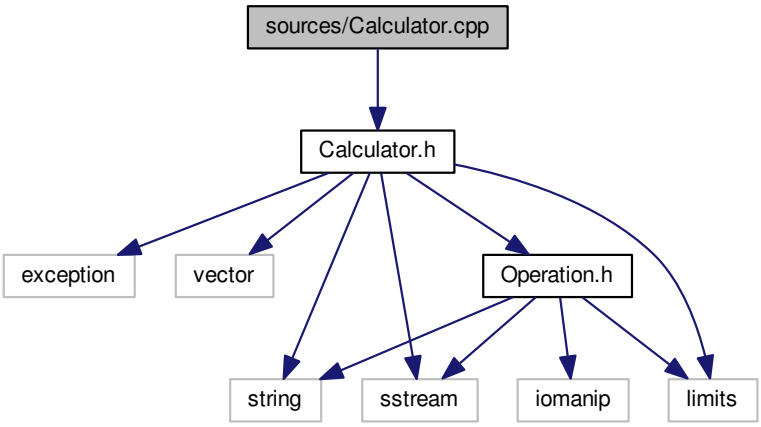
Definition at line 47 of file main.cpp.

6.4 sources/Calculator.cpp File Reference

This file holds the definitions for the Calculator class.

```
#include <Calculator.h>
```

Include dependency graph for Calculator.cpp:



6.4.1 Detailed Description

This file holds the definitions for the Calculator class.

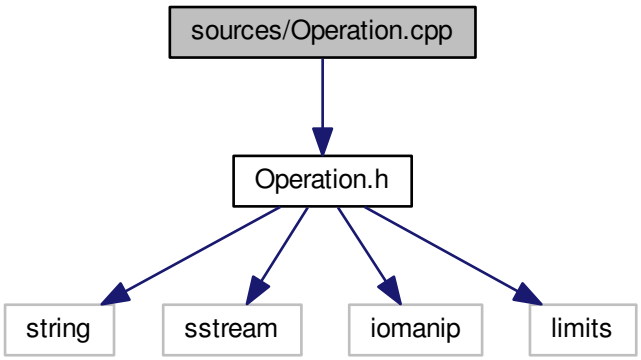
Definition in file Calculator.cpp.

6.5 sources/Operation.cpp File Reference

This file holds the definitions for the Operation class.

```
#include <Operation.h>
```

Include dependency graph for Operation.cpp:



6.5.1 Detailed Description

This file holds the definitions for the Operation class.

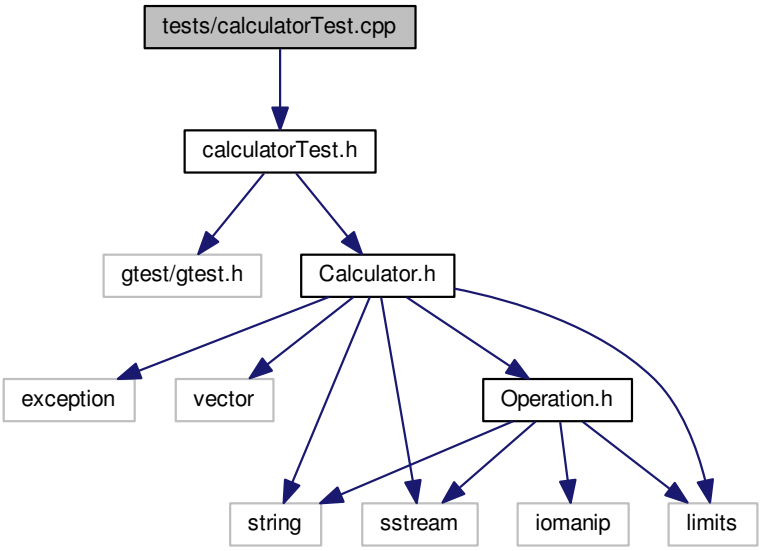
Definition in file Operation.cpp.

6.6 tests/calculatorTest.cpp File Reference

This file holds all of the tests within the CalculatorTest fixture.

```
#include "calculatorTest.h"
```

Include dependency graph for calculatorTest.cpp:



Functions

- TEST_F (CalculatorTest, givenInitialValueCalculatorHoldsValue)

  Tests if the calculator holds the given value.
- TEST_F (CalculatorTest, givenInitialValueWhenAddingValueCalculatorCorrectlySums)

  Tests if the calculator correctly sums.
- TEST_F (CalculatorTest, givenInitialValueWhenAddingValueCalculatorCorrectlySumsInts)

  Tests if the calculator correctly sums.
- TEST_F (CalculatorTest, givenInitialValueWhenSubtractingValueCalculatorCorrectlySubtracts)

  Tests if the calculator correctly subtracts.
- TEST_F (CalculatorTest, givenInitialValueWhenSubtractingValueCalculatorCorrectlySubtractsInts)

  Tests if the calculator correctly subtracts.
- TEST_F (CalculatorTest, givenInitialValueWhenMultiplyingValueCalculatorCorrectlyMultiplies)

  Tests if the calculator correctly multiplies.
- TEST_F (CalculatorTest, givenInitialValueWhenMultiplyingValueCalculatorCorrectlyMultipliesInts)

  Tests if the calculator correctly multiplies.
- TEST_F (CalculatorTest, givenInitialZeroAsValueWhenMultiplyingValueCalculatorCorrectlyMultipliesBy0)

  Tests if the calculator correctly multiplies by 0.
- TEST_F (CalculatorTest, givenInitialValueWhenDividingValueCalculatorCorrectlyDivides)

  Tests if the calculator correctly divides.
- TEST_F (CalculatorTest, givenInitialValueWhenDividingByZeroCalculatorThrowsAnException)

  Tests if the calculator throws exception when dividing by zero.
- TEST_F (CalculatorTest, givenInitialValueAfterPerformingOperationsCalculatorHoldsItsPreviousValue)

  Tests if the calculator correctly gives the previous totals before the last operation.
- TEST_F (CalculatorTest, givenInitialValueAfterPerformingOperationsCalculatorHoldsItsPreviousValue↩ WhenDividingByZero)

Tests if the calculator reverts to the correct value after dividing by zero.

- TEST_F (CalculatorTest, afterPerformingOperationsAndUndoingOneTheCurrentTotalIsCorrect)

  Tests if the calculator reverts to the correct value after undoing one of many operations.

- TEST_F (CalculatorTest, givenInitialValueAfterPerformingOperationsCorrectlyUndoes)

  Tests if the calculator correctly undoes consecutive operations.

- TEST_F (CalculatorTest, givenInitialValueWhenNoOperationsPerformedUndoDoesNothing)

  Tests if the calculator handles undoing, after no operations have been performed.

- TEST_F (CalculatorTest, givenInitialValueAfterPerformingOperationsReturnsCorrectString)

  Tests if the calculator correctly undoes consecutive operations with a division by zero among them.

- TEST_F (CalculatorTest, afterPerformingOperationsWhenClearIssuedCalculatorCleared)

  Tests if the calculator correctly resets.

- TEST_F (CalculatorTest, whenNoOperationsPerformedGetPreviousValueReturnsCurrentValue)

  Tests if the calculator correctly returns the current value as previous, when no operations have been performed.

6.6.1 Detailed Description

This file holds all of the tests within the CalculatorTest fixture.

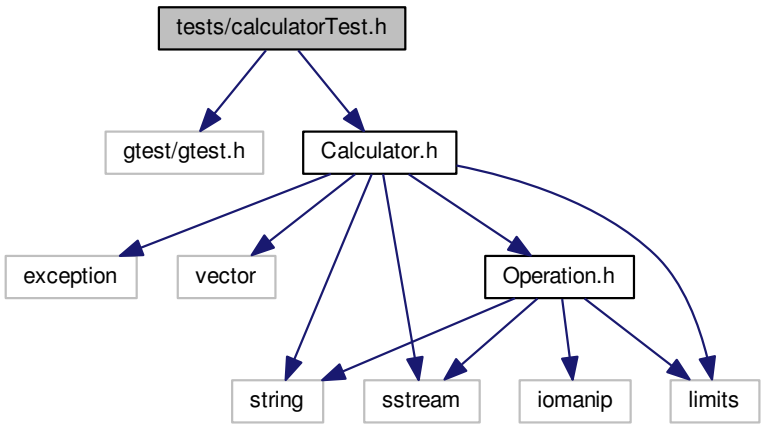
Definition in file calculatorTest.cpp.

6.7 tests/calculatorTest.h File Reference

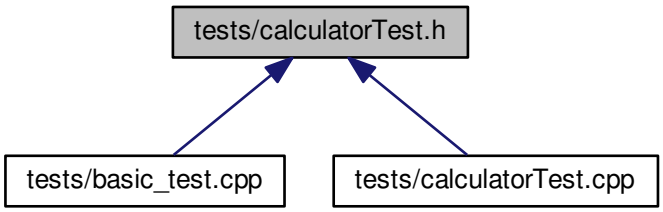
Holds the CalculatorTest fixture class.

```
#include <gtest/gtest.h>
#include <Calculator.h>
```

Include dependency graph for calculatorTest.h:



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



Classes

- class CalculatorTest

This class is a test fixture, that sets up the necessary data before performing tests of the Calculator class.

6.7.1 Detailed Description

Holds the CalculatorTest fixture class.

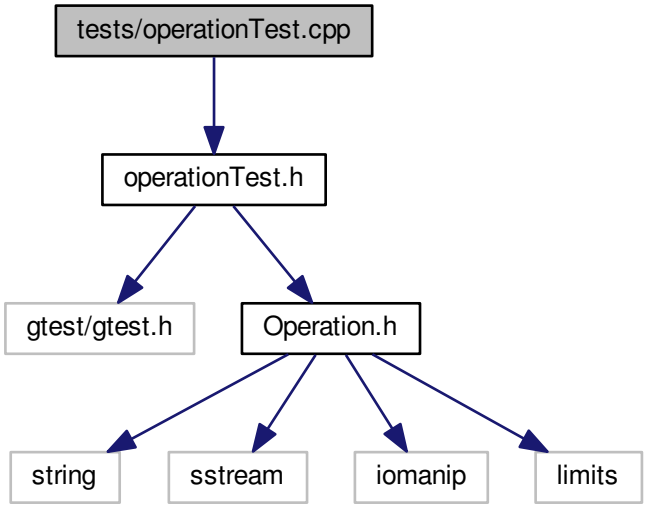
Definition in file calculatorTest.h.

6.8 tests/operationTest.cpp File Reference

This file holds all of the tests within the CalculatorTest fixture.

```
#include "operationTest.h"
```

Include dependency graph for operationTest.cpp:



Functions

- TEST (OperationTest, givenZeroValueOperationProperlySums)

  Tests if the operation correctly adds values.
- TEST (OperationTest, givenZeroValueOperationProperlyUndoesSum)

  Tests if the operation correctly undoes addition.
- TEST (OperationTest, givenZeroValueOperationProperlySubtracts)

  Tests if the operation correctly subtracts values.
- TEST (OperationTest, givenZeroValueOperationProperlyUndoesSubtraction)

  Tests if the operation correctly undoes subtraction.
- TEST (OperationTest, givenZeroValueOperationProperlyMultiplies)

  Tests if the operation correctly multiplies by a value.
- TEST (OperationTest, givenZeroValueOperationProperlyUndoesMultiplication)

  Tests if the operation correctly undoes multiplication.
- TEST (OperationTest, givenZeroValueOperationProperlyDivides)

  Tests if the operation correctly divides by a value.
- TEST (OperationTest, givenZeroValueOperationProperlyUndoesDivision)

  Tests if the operation correctly undoes division.
- TEST (OperationTest, givenZeroValueOperationIgnoresDividingByZero)

  Tests if the operation returns the given value when trying to divide by zero.
- TEST (OperationTest, givenZeroValueOperationProperlyUndoesDivisionByZero)

  Tests if the operation correctly undoes division by zero.
- TEST (OperationTest, givenAnOperationWhenConvertingToStringBehavesCorrectly)

  Tests if the operation correctly converts the opreator and value to string.

6.8.1 Detailed Description

This file holds all of the tests within the CalculatorTest fixture.

Definition in file operationTest.cpp.

Index