:test-result: PASS # A test name that starts with a #

:test-result: PASS #542

:test-result: PASS #809

:test-result: FAIL 'Not' checks that should fail

:test-result: PASS 'Not' checks that should succeed

:test-result: PASS (unimplemented) static bools can be evaluated

:test-result: FAIL A METHOD\_AS\_TEST\_CASE based test run that fails

:test-result: PASS A METHOD\_AS\_TEST\_CASE based test run that succeeds

:test-result: FAIL A TEST\_CASE\_METHOD based test run that fails

:test-result: PASS A TEST\_CASE\_METHOD based test run that succeeds

:test-result: FAIL A couple of nested sections followed by a failure

:test-result: FAIL A failing expression with a non streamable type is still captured

:test-result: PASS AllOf matcher

:test-result: PASS An empty test with no assertions

:test-result: PASS An expression with side-effects should only be evaluated once

:test-result: FAIL An unchecked exception reports the line of the last assertion

:test-result: PASS Anonymous test case 1

:test-result: PASS AnyOf matcher

:test-result: PASS Approximate PI

:test-result: PASS Approximate comparisons with different epsilons

:test-result: PASS Approximate comparisons with floats

:test-result: PASS Approximate comparisons with ints

:test-result: PASS Approximate comparisons with mixed numeric types

:test-result: PASS Assertions then sections

:test-result: PASS Character pretty printing

:test-result: PASS Comparing function pointers

:test-result: PASS Comparing member function pointers

:test-result: PASS Comparisons between ints where one side is computed

:test-result: PASS Comparisons between unsigned ints and negative signed ints match c++ standard behaviour

:test-result: PASS Comparisons with int literals don't warn when mixing signed/ unsigned

:test-result: FAIL Contains string matcher

:test-result: FAIL Custom exceptions can be translated when testing for nothrow

:test-result: FAIL Custom exceptions can be translated when testing for throwing as something else

:test-result: FAIL Custom std-exceptions can be custom translated

:test-result: PASS Demonstrate that a non-const == is not used

:test-result: FAIL EndsWith string matcher

:test-result: XFAIL Equality checks that should fail

:test-result: PASS Equality checks that should succeed

:test-result: PASS Equals

:test-result: FAIL Equals string matcher

:test-result: PASS Exception messages can be tested for

:test-result: FAIL Expected exceptions that don't throw or unexpected exceptions fail the test

:test-result: FAIL FAIL aborts the test

:test-result: FAIL FAIL does not require an argument

:test-result: PASS Factorials are computed

:test-result: PASS Generator over a range of pairs

:test-result: PASS Generators over two ranges

:test-result: PASS Greater-than inequalities with different epsilons

:test-result: PASS INFO and WARN do not abort tests

:test-result: FAIL INFO gets logged on failure

:test-result: FAIL INFO gets logged on failure, even if captured before successful assertions

:test-result: XFAIL Inequality checks that should fail

:test-result: PASS Inequality checks that should succeed

:test-result: PASS Less-than inequalities with different epsilons

:test-result: PASS Long strings can be wrapped

:test-result: PASS Long text is truncted

:test-result: PASS ManuallyRegistered

:test-result: PASS Matchers can be (AllOf) composed with the && operator

:test-result: PASS Matchers can be (AnyOf) composed with the || operator

:test-result: PASS Matchers can be composed with both && and ||

:test-result: FAIL Matchers can be composed with both && and || - failing

:test-result: PASS Matchers can be negated (Not) with the ! operator

:test-result: FAIL Matchers can be negated (Not) with the ! operator - failing

:test-result: FAIL Mismatching exception messages failing the test

:test-result: PASS Nice descriptive name

:test-result: FAIL Non-std exceptions can be translated

:test-result: PASS NotImplemented exception

:test-result: PASS Objects that evaluated in boolean contexts can be checked

:test-result: PASS Operators at different namespace levels not hijacked by Koenig lookup

:test-result: FAIL Ordering comparison checks that should fail

:test-result: PASS Ordering comparison checks that should succeed

:test-result: FAIL Output from all sections is reported

:test-result: PASS Parse test names and tags

:test-result: PASS Parsing a std::pair

:test-result: PASS Pointers can be compared to null

:test-result: PASS Pointers can be converted to strings

:test-result: PASS Process can be configured on command line

:test-result: FAIL SCOPED\_INFO is reset for each loop

:test-result: PASS SUCCEED counts as a test pass

:test-result: PASS SUCCESS does not require an argument

:test-result: PASS Scenario: BDD tests requiring Fixtures to provide commonly-accessed data or methods

:test-result: PASS Scenario: Do that thing with the thing

:test-result: PASS Scenario: This is a really long scenario name to see how the list command deals with wrapping

:test-result: PASS Scenario: Vector resizing affects size and capacity

A string sent directly to stdout

A string sent directly to stderr

:test-result: PASS Sends stuff to stdout and stderr

:test-result: PASS Some simple comparisons between doubles

Message from section one

Message from section two

:test-result: PASS Standard output from all sections is reported

:test-result: FAIL StartsWith string matcher

:test-result: PASS String matchers

hello

hello

:test-result: PASS Strings can be rendered with colour

:test-result: FAIL Tabs and newlines show in output

:test-result: PASS Tag alias can be registered against tag patterns

:test-result: PASS Test case with one argument

:test-result: PASS Test enum bit values

:test-result: PASS Text can be formatted using the Text class

:test-result: PASS The NO\_FAIL macro reports a failure but does not fail the test

:test-result: FAIL This test 'should' fail but doesn't

:test-result: PASS Tracker

:test-result: FAIL Unexpected exceptions can be translated

:test-result: PASS Use a custom approx

:test-result: PASS Variadic macros

:test-result: PASS When checked exceptions are thrown they can be expected or unexpected

:test-result: FAIL When unchecked exceptions are thrown directly they are always failures

:test-result: FAIL When unchecked exceptions are thrown during a CHECK the test should continue

:test-result: FAIL When unchecked exceptions are thrown during a REQUIRE the test should abort fail

:test-result: FAIL When unchecked exceptions are thrown from functions they are always failures

:test-result: FAIL When unchecked exceptions are thrown from sections they are always failures

:test-result: PASS When unchecked exceptions are thrown, but caught, they do not affect the test

:test-result: PASS Where the LHS is not a simple value

:test-result: PASS Where there is more to the expression after the RHS

:test-result: PASS X/level/0/a

:test-result: PASS X/level/0/b

:test-result: PASS X/level/1/a

:test-result: PASS X/level/1/b

:test-result: PASS XmlEncode

:test-result: PASS atomic if

:test-result: PASS boolean member

:test-result: PASS checkedElse

:test-result: FAIL checkedElse, failing

:test-result: PASS checkedIf

:test-result: FAIL checkedIf, failing

:test-result: PASS comparisons between const int variables

:test-result: PASS comparisons between int variables

:test-result: PASS even more nested SECTION tests

:test-result: PASS first tag

spanner:test-result: PASS has printf

:test-result: FAIL just failure

:test-result: PASS just info

:test-result: FAIL looped SECTION tests

:test-result: FAIL looped tests

:test-result: FAIL more nested SECTION tests

:test-result: PASS nested SECTION tests

:test-result: PASS non streamable - with conv. op

:test-result: PASS not allowed

:test-result: PASS null strings

:test-result: PASS pair<pair<int,const char \*,pair<std::string,int> > -> toString

:test-result: PASS pointer to class

:test-result: PASS random SECTION tests

:test-result: PASS replaceInPlace

:test-result: PASS second tag

:test-result: FAIL send a single char to INFO

:test-result: FAIL sends information to INFO

:test-result: PASS std::pair<int,const std::string> -> toString

:test-result: PASS std::pair<int,std::string> -> toString

:test-result: PASS std::vector<std::pair<std::string,int> > -> toString

:test-result: FAIL string literals of different sizes can be compared

:test-result: PASS toString on const wchar\_t const pointer returns the string contents

:test-result: PASS toString on const wchar\_t pointer returns the string contents

:test-result: PASS toString on wchar\_t const pointer returns the string contents

:test-result: PASS toString on wchar\_t returns the string contents

:test-result: PASS toString( has\_maker )

:test-result: PASS toString( has\_maker\_and\_toString )

:test-result: PASS toString( has\_toString )

:test-result: PASS toString( vectors<has\_maker )

:test-result: SKIP toString( vectors<has\_maker\_and\_toString )

:test-result: SKIP toString( vectors<has\_toString )

:test-result: PASS toString(enum w/operator<<)

:test-result: PASS toString(enum)

:test-result: PASS vector<int> -> toString

:test-result: PASS vector<string> -> toString

:test-result: PASS vectors can be sized and resized

:test-result: PASS xmlentitycheck