What Are We Covering?

- Google Test
  - Google Test Macros
  - Test **Fixtures**
  - Manipulating Tests
  - Advanced Topics
    - Sharing Test Logic
    - Typed Test
    - Type-Parameterized Test
    - Assertion in Subroutines
    - Testing Private Code

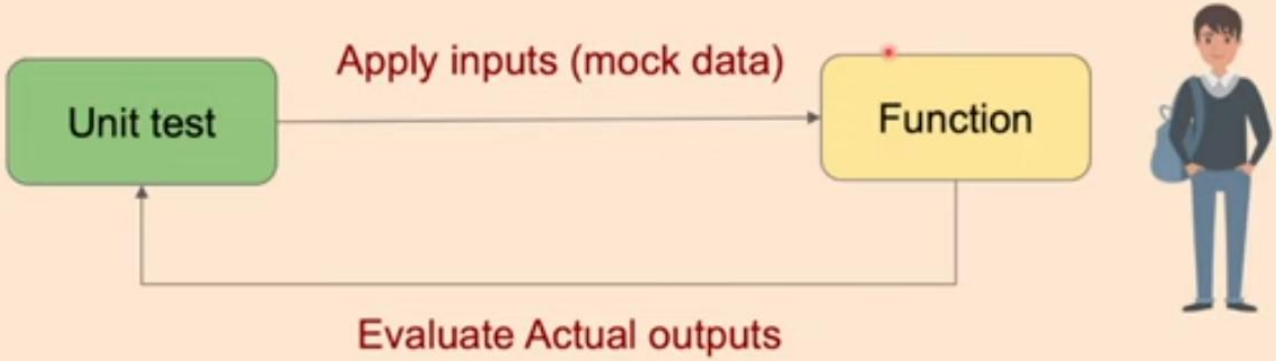
- Google Mock
  - Matchers
  - **EXPECT_THAT**
  - Mocking Classes and Functions
    - **EXPECT_CALL**
    - **ON_CALL**
  - Advanced Topics
    - Forcing **Sequence**
    - Checkpoints
    - Delegation
    - GMock Custom Matchers

Unit Testing

Suggested: Backtracking (Job Interview Special)

Test your code. Leave nothing to luck!

- (Almost) every sane programmer



A unit test is a piece of code that tests a function or a class.

Why Unit Testing?

🔨 Shows code **works as intended**

```
TEST(DivideTest, SimpleDivision) {
  EXPECT_EQ(Divide(6, 2), 3);
  EXPECT_EQ(Divide(2, 2), 1);
}

TEST(DivideTest, ThrowsForDivideByZero) {
  EXPECT_THROW(Divide(6, 0), DivideByZero);
}
```

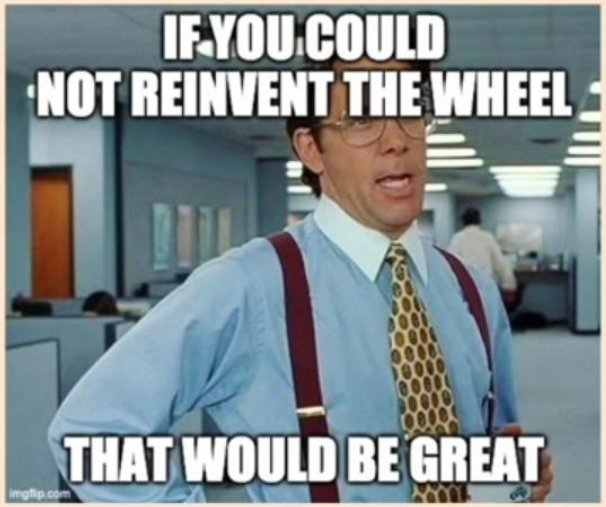
📄 Provides **documentation**

🧘 Ensures **changes** don't **break** things

- Simplify refactoring

🐛 Helps with finding **bugs**

Why a Test Platform?

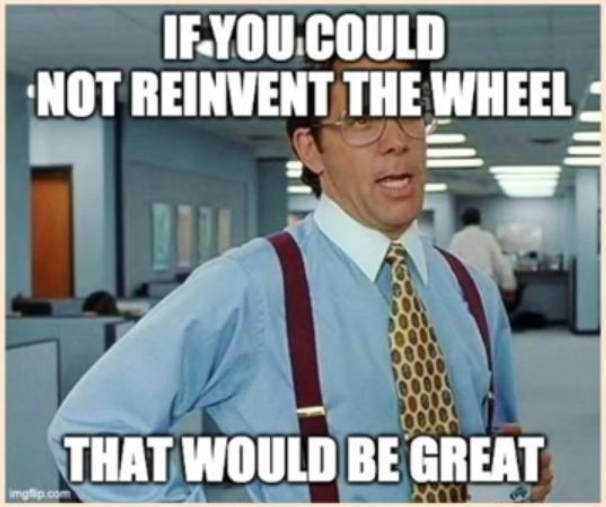



```
template <class T>
int ExpectEqual(T expected, T actual) {
  if (expected == actual) {
    return 0;
  } else {
    return 1;
  }
}
int main() {
  int result = 0;
  result += ExpectEqual(Fib(0), 0);
  result += ExpectEqual(Fib(1), 1);
  result += ExpectEqual(Fib(2), 1);
  result += ExpectEqual(Fib(3), 2);

  std::cout <<
    (result == 0 ? "PASS" : "FAIL") <<
    std::endl;
  return 0;
}
```

```
TEST(FibTest, BasicTest) {
  EXPECT_EQ(Fib(0), 0);
  EXPECT_EQ(Fib(1), 1);
  EXPECT_EQ(Fib(2), 1);
  EXPECT_EQ(Fib(3), 2);
  EXPECT_EQ(Fib(4), 3);
}
```

Because don't reinvent the wheel!

Why a Test Platform?



Because don't reinvent the wheel!

```
template <class T>
int ExpectEqual(T expected, T actual) {
  if (expected == actual) {
    return 0;
  } else {
    return 1;
  }
}
int main() {
  int result = 0;
  result += ExpectEqual(Fib(0), 0);
  result += ExpectEqual(Fib(1), 1);
  result += ExpectEqual(Fib(2), 1);
  result += ExpectEqual(Fib(3), 2);

  std::cout <<
    (result == 0 ? "PASS" : "FAIL") <<
    std::endl;
  return 0;
}
```

```
TEST(FibTest, BasicTest) {
  EXPECT_EQ(Fib(0), 0);
  EXPECT_EQ(Fib(1), 1);
  EXPECT_EQ(Fib(2), 1);
  EXPECT_EQ(Fib(3), 2);
  EXPECT_EQ(Fib(4), 3);
}
```

```
TEST(AtmMachine, CanWithdrawSimple) {
  // Arrange

  MockBankServer mock_bankserver;

  // Expectations
  EXPECT_CALL(mock_bankserver,
          GetBalance(1234))
              .Times(1)
              .WillOnce(Return(2000));
  // Act
  AtmMachine atm_machine(&mock_bankserver);
  bool withdraw_result = atm_machine.Withdraw(1234, 1000);
  // Assert
  EXPECT_TRUE(withdraw_result);
}
```

Settings

Quick Definitions

Test Suite:

A group of logically **related tests** cases.

E.g. **different inputs** for a function.

Fixture:

A Class factoring out **common code** for multiple cases.

E.g. **Setup** and **TearDown**.

Mock Classes/Objects:

Simplified and lightweight classes to untangle complex **dependencies** while testing.

Settings

Test Framework

```
TEST(FibTest, NegativeValues) {
  EXPECT_LT(Fib(-1), 0);
  EXPECT_LT(Fib(-10), 0);
  EXPECT_THAT(Fib(-10), Lt(0));
}
```

```
class QueueTest : public ::testing::Test {
  void SetUp() override {
    ...
  }

  void TearDown() override {
    ...
  }
};
```

```
tests/gtest_demo/assert_vs_expect.cc:16: Failure
Value of: false
  Actual: false
Expected: true
[  FAILED  ] AssertAndExpect.ExpectFailIsNotFatal (0 ms)
[----------] 2 tests from AssertAndExpect (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 0 tests.
[  FAILED  ] 2 tests, listed below:
[  FAILED  ] AssertAndExpect.AssertFailIsFatal
[  FAILED  ] AssertAndExpect.ExpectFailIsNotFatal

 2 FAILED TESTS
```

Automate the execution

E.g. implementing main()

Assertions, **Checks**, and **Matchers**

Test **Organization**

E.g.: Suites, Fixtures

Reports

Settings

Test Framework

```
TEST(FibTest, NegativeValues) {
  EXPECT_LT(Fib(-1), 0);
  EXPECT_LT(Fib(-10), 0);
  EXPECT_THAT(Fib(-10), Lt(0));
}
```

```
class QueueTest : public ::testing::Test {
  void SetUp() override {
    ...
  }

  void TearDown() override {
    ...
  }
};
```

```
tests/gtest_demo/assert_vs_expect.cc:16: Failure
Value of: false
  Actual: false
Expected: true
[  FAILED  ] AssertAndExpect.ExpectFailIsNotFatal (0 ms)
[----------] 2 tests from AssertAndExpect (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 0 tests.
[  FAILED  ] 2 tests, listed below:
[  FAILED  ] AssertAndExpect.AssertFailIsFatal
[  FAILED  ] AssertAndExpect.ExpectFailIsNotFatal

 2 FAILED TESTS
```

Automate the execution

E.g. implementing main()

Assertions, **Checks**, and **Matchers**

Test **Organization**

E.g.: Suites, Fixtures

Reports

Google Test and Mock

Catch2

Doctest

Boost.Test

Settings

Test Frameworks

	Google Test/Mock	Catch2	Boost.Test	Doctest
Availability	**Requires building**	**Header only**	**Header only, static library, or shared library**	**Header only**
Mocks	**Direct support**	**No direct support**	**No direct support**	**No direct support**
Assertions and checks	Yes	Yes	Yes	Yes
Suites	Yes	Yes	Yes	Yes
Fixtures	Yes	Yes	Yes	Yes

Settings

Prerequisite: Installing Bazel

This repo uses `Bazel` for building C++ files. You can install Bazel using this link.

Cloning this repo

```
git clone https://github.com/ourarash/cpp-template.git
```

Examples:

Hello World Example:

You can run this using `bazel`:

```
bazel run src/main:main
```

https://github.com/ourarash/cpp-template

Our First Unit Test

Our First Unit Test

```
// Calculates the n'th Fibonacci number.
// Returns a negative value if
// the input is negative.
int Fib(int n) {
  if (n <= 1) {
    return n;
  }
  return Fib(n - 1) + Fib(n - 2);
}
```

Suite name

Test name

```
#include "gtest/gtest.h"

TEST(FibTest, NonNegativeValues) {
  EXPECT_EQ(Fib(0), 0);
  EXPECT_EQ(Fib(1), 1);
  EXPECT_EQ(Fib(5), 5);
  EXPECT_EQ(Fib(6), 8);
}

TEST(FibTest, NegativeValues) {
  EXPECT_LT(Fib(-1), 0);
  EXPECT_LT(Fib(-10), 0);
}
```

Valid **C++ identifiers**:

- No **underscores** (_).
- Test's *full name* is:
  - `suite_name.test_name`
- Tests from different test suites can have the same individual name.

```
tests/gtest_demo/fib_test.cc
bazel test tests/gtest_demo:fib_test
```

```
ari@Arios-MacBook-Pro:~/github/cpp-template$ bazel test //tests/gtest_demo:fib_test
INFO: Analyzed target //tests/gtest_demo:fib_test (23 packages loaded, 523 targets configured).
INFO: Found 1 test target...
Target //tests/gtest_demo:fib_test up-to-date:
  bazel-bin/tests/gtest_demo/fib_test
INFO: Elapsed time: 11.185s, Critical Path: 3.66s
INFO: 25 processes: 6 internal, 19 darwin-sandbox.
INFO: Build completed successfully, 25 total actions
//tests/gtest_demo:fib_test                                              PASSED in 0.4s

Executed 1 out of 1 test: 1 test passes.
INFO: Build completed successfully, 25 total actions
ari@Arios-MacBook-Pro:~/github/cpp-template$
```

```
TEST(FibTest, NonNegativeValues) {
  EXPECT_EQ(Fib(0), 0);
  EXPECT_EQ(Fib(1), 1);
  EXPECT_EQ(Fib(5), 5);
  EXPECT_EQ(Fib(6), 8);
}

TEST(FibTest, NegativeValues) {
  EXPECT_LT(Fib(-1), 0);
  EXPECT_LT(Fib(-10), 0);
}
```

```
exec ${PAGER:-/usr/bin/less} "$0" || exit 1
Executing tests from //tests:fib_test
-----------------------------------------------------------------------------
Running main() from gmock_main.cc
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from FibTest
[ RUN      ] FibTest.NonNegativeValues
[       OK ] FibTest.NonNegativeValues (0 ms)
[ RUN      ] FibTest.NegativeValues
[       OK ] FibTest.NegativeValues (0 ms)
[----------] 2 tests from FibTest (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 2 tests.
```

```
std::string GetHello(std::string_view in) {
  if (in.size() == 0) {
    return std::string("hello, world");
  } else {
    return std::string("Hello, ") +   in.data();
  }
}
```

```
TEST(GetHello, CanMakeHelloWorldWithEmptyInput) {
  EXPECT_EQ(GetHello(""), "hello, world");
}

TEST(GetHello, CanMakeHelloWithNonEmptyInput) {
  EXPECT_EQ(GetHello("Ari"), "Hello, Ari");
}
```

```
tests/gtest_demo/hello_world_test.cc
> bazel test //tests/gtest_demo:hello_world_test
```

tests/gtest_demo/BUILD

```
cc_test(
  name = "hello_world_test",
  srcs = ["hello_world_test.cc"],
  deps = [
    "@com_google_googletest//:gtest_main",
  ],
)
```

```
cc_binary(
  name = "hello_world_test_main",
  srcs = ["hello_world_test_main.cc"],
  deps = [
    "@com_google_googletest//:gtest",
  ],
)
```

```
TEST(GetHello, CanMakeHelloWorldWithEmptyInput) {
  EXPECT_EQ(GetHello(""), "hello, world");
}

TEST(GetHello, CanMakeHelloWithNonEmptyInput) {
  EXPECT_EQ(GetHello("Ari"), "Hello, Ari");
}
```

```
TEST(GetHello, CanMakeHelloWorldWithEmptyInput) {
  EXPECT_EQ(GetHello(""), "hello, world");
}

TEST(GetHello, CanMakeHelloWithNonEmptyInput) {
  EXPECT_EQ(GetHello("Ari"), "Hello, Ari");
}

int main(int argc, char** argv) {
  testing::InitGoogleTest(&argc, argv);
  return RUN_ALL_TESTS();
}
```

```
tests/gtest_demo/hello_world_test.cc
> bazel test //tests/gtest_demo:hello_world_test
```

```
tests/gtest_demo/hello_world_test_main.cc
> bazel run //tests/gtest_demo:hello_world_test_main
```

Google Test Assertion Macros

ASSERT vs EXPECT

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_TRUE(condition)	EXPECT_TRUE(condition)	condition is true
ASSERT_FALSE(condition)	EXPECT_FALSE(condition)	condition is false

```
TEST(AssertAndExpect, AssertFailIsFatal) {
  ASSERT_TRUE(false);
  std::cout << "This line won't be executed!" << std::endl;
  EXPECT_TRUE(false);
}
```

```
TEST(AssertAndExpect, ExpectFailIsNotFatal) {
  EXPECT_TRUE(false);
  std::cout << "This line will be executed!" << std::endl;
  EXPECT_TRUE(false);
}
```

```
tests/gtest_demo/assert_vs_expect.cc:14: Failure
Value of: false
  Actual: false
Expected: true
This line will be executed!
tests/gtest_demo/assert_vs_expect.cc:16: Failure
Value of: false
  Actual: false
Expected: true
```

tests/gtest_demo/assert_vs_expect.cc

Binary Comparison Assertions

```
TEST(FibTest, NonNegativeValues) {
  EXPECT_EQ(Fib(0), 0);
}
```

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_EQ(val1, val2);	EXPECT_EQ(val1, val2);	val1 == val2
ASSERT_NE(val1, val2);	EXPECT_NE(val1, val2);	val1 != val2
ASSERT_LT(val1, val2);	EXPECT_LT(val1, val2);	val1 < val2
ASSERT_LE(val1, val2);	EXPECT_LE(val1, val2);	val1 <= val2
ASSERT_GT(val1, val2);	EXPECT_GT(val1, val2);	val1 > val2
ASSERT_GE(val1, val2);	EXPECT_GE(val1, val2);	val1 >= val2

- Built-in types
- Types that overload these operators

C Strings

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_STREQ(expected, actual)	EXPECT_STREQ(expected, actual)	the two C strings have identical content
ASSERT_STRNE(str1, str2)	EXPECT_STRNE(str1, str2)	the two C strings have different content
ASSERT_STRCASEEQ(exp, act)	EXPECT_STRCASEEQ(exp, act)	the two C strings have the same content, ignoring case
ASSERT_STRCASENE(str1, str2)	EXPECT_STRCASENE(str1, str2)	the two C strings have different content, ignoring case.

- To compare two string objects, use EXPECT_EQ or EXPECT_NE instead.
- To compare a C string with NULL, use:
  EXPECT_EQ(*c_string*, nullptr)
  EXPECT_NE(*c_string*, nullptr).

```
const char kString1[] = "String 1";
const char kString11[] = "STRING 1";
const char kString2[] = "String 2";
//-----------------------------------------------------
TEST(CStringTest, SimpleComparisons) {
  EXPECT_STREQ(kString1, "String 1");
  EXPECT_STRNE(kString1, kString2);

  EXPECT_STRNE(kString1, kString11);
  EXPECT_STRCASEEQ(kString1, kString1);
}
```

tests/gtest_demo/c_string_assertions.cc

Floating points

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_FLOAT_EQ(exp, act)	EXPECT_FLOAT_EQ(exp, act)	The two float values are almost equal
ASSERT_DOUBLE_EQ(exp, act)	EXPECT_DOUBLE_EQ(exp, act)	The two double values are almost equal
ASSERT_NEAR(val1, val2, abs_err)	EXPECT_NEAR(val1, val2, abs_err)	The difference between the val1 and val2 doesn't exceed abs_err.

* By “almost equal” we mean the two values are within 4 ULP’s from each other.

```
const float a = 67329.234;
const float b = 67329.242; // Exactly 1 ULP away from a
TEST(FloatingPointTest, SimpleAssertion) {
  EXPECT_EQ(a, b); // Exactly 1 ULP away, Fails!
}

TEST(FloatingPointTest, FpAssertion) {
  EXPECT_FLOAT_EQ(a, b); // Exactly 1 ULP away, Passes
}

TEST(FloatingPointTest, FpAssertionNear) {
  EXPECT_NEAR(a, b, 0.2); // 0.2 Difference
}
```

tests/gtest_demo/floating_points.cc

Predicates

```
bool IsAlmostEq(int a, int b) {
  if (std::abs(a - b) < 10)
    return true;
  else
    return false;
}

TEST(ExpectPredTest, IsAlmostEq) {
  int a = 0;
  int b = 1;
  EXPECT_TRUE(IsAlmostEq(a, b));
  EXPECT_PRED2(IsAlmostEq, a, b);
}
```

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_PRED1(pred, val1)	EXPECT_PRED1(pred, val1)	pred(val1) returns true
ASSERT_PRED2(pred, val1, val2)	EXPECT_PRED2(pred, val1, val2	pred(val1, val2) returns true
...	...	...

tests/gtest_demo/predicate.cc

Predicates

```
bool IsAlmostEq(int a, int b) {
  if (std::abs(a - b) < 10)
    return true;
  else
    return false;
}

TEST(ExpectPredTest, IsAlmostEq) {
  int a = 0;
  int b = 1;
  EXPECT_TRUE(IsAlmostEq(a, b));
  EXPECT_PRED2(IsAlmostEq, a, b);
}
```

```
AssertionResult IsAlmostEq2(int a, int b) {
  if (std::abs(a - b) < 10) {
    return AssertionSuccess();
  } else {
    return AssertionFailure()
      << "a is " << a << " and b is: " << b
      << ". The difference is: " << std::abs(a - b);
  }
}


TEST(ExpectPredTest, IsAlmostEq2) {
  int a = 10;
  int b = 1000;
  EXPECT_TRUE(IsAlmostEq2(a, b));
}
```

```
[ RUN      ] ExpectPredTest.IsAlmostEq
tests/gtest_demo/predicate.cc:26: Failure
Value of: IsAlmostEq(a, b)
  Actual: false
Expected: true
tests/gtest_demo/predicate.cc:27: Failure
IsAlmostEq(a, b) evaluates to false, where
a evaluates to 10
b evaluates to 1000
[  FAILED  ] ExpectPredTest.IsAlmostEq (0 ms)
[ RUN      ] ExpectPredTest.IsAlmostEq2
```

```
[ RUN      ] ExpectPredTest.IsAlmostEq2
tests/gtest_demo/predicate.cc:33: Failure
Value of: IsAlmostEq2(a, b)
  Actual: false (a is 10 and b is: 1000. The difference is: 990)
Expected: true
[  FAILED  ] ExpectPredTest.IsAlmostEq2 (0 ms)
```

tests/gtest_demo/predicate.cc

Exceptions

Fatal assertion	Nonfatal assertion	Verifies
ASSERT_THROW(stmt, exc_type)	EXPECT_THROW(stmt, exc_type)	stmt throws an exception of the given exc_type.
ASSERT_ANY_THROW(stmt)	EXPECT_ANY_THROW(stmt)	stmt throws an exception of any type.
ASSERT_NO_THROW(stmt)	EXPECT_NO_THROW(stmt)	stmt throws no exception

```
int Divide(int a, int b) {
  if (b == 0) {
    throw DivideByZero();
  }
  return a / b;
}

TEST(DivideTest, ThrowsForDivideByZero) {
  EXPECT_THROW(Divide(6, 0), DivideByZero);
}

TEST(DivideTest, DoesNotThrowForDivideByNonZero) {
  EXPECT_NO_THROW(Divide(6, 2));
}
```

tests/gtest_demo/exceptions.cc

Exceptions

```
class DivideByZero : public virtual std::exception {
 public:
  const char* what() const noexcept override {
    return "Divide By Zero Error!";
  }
};

int Divide(int a, int b) {
  if (b == 0) {
    throw DivideByZero();
  }
  return a / b;
}
```

```
TEST(DivideTest, ThrowsForDivideByZero) {
  EXPECT_THROW(Divide(6, 0), DivideByZero);
  EXPECT_NO_THROW(Divide(6, 2));
}

TEST(DivideTest, DoesNotThrowForDivideByNonZero) {
  EXPECT_NO_THROW(Divide(6, 2));
}
```

Doesn't check the exception's message

```
TEST(DivideTest, ThrowsCorrectMessageForDivideByZero) {
  EXPECT_THROW(
    {
      try {
        Divide(6, 0);
      } catch (const DivideByZero& e) {
        EXPECT_STREQ("Divide By Zero Error!", e.what());
        throw;
      }
    },
    DivideByZero);
}
```

Checks both exception and the message

tests/gtest_demo/exceptions.cc

Death Test

EXPECT_DEATH

```
EXPECT_DEATH( statement , matcher )
ASSERT_DEATH( statement , matcher )
```

Verifies that `statement` causes the process to terminate with a nonzero exit status and produces `stderr` output that matches `matcher`.

EXPECT_DEATH_IF_SUPPORTED

```
EXPECT_DEATH_IF_SUPPORTED( statement , matcher )
ASSERT_DEATH_IF_SUPPORTED( statement , matcher )
```

If death tests are supported, behaves the same as `EXPECT_DEATH`. Otherwise, verifies nothing.

EXPECT_EXIT

```
EXPECT_EXIT( statement , predicate , matcher )
ASSERT_EXIT( statement , predicate , matcher )
```

Verifies that `statement` causes the process to terminate with an exit status that satisfies `predicate`, and produces `stderr` output that matches `matcher`.

The parameter `predicate` is a function or functor that accepts an `int` exit status and returns a `bool`. GoogleTest provides two predicates to handle common cases:

```
// Returns true if the program exited normally with the given exit status code.
::testing::ExitedWithCode(exit_code);

// Returns true if the program was killed by the given signal.
// Not available on Windows.
::testing::KilledBySignal(signal_number);
```

Google Test Macros – Death Test

```
void Exit(int exitCode = 0) {
  // Exit with the given code.
  std::exit(exitCode);
}

void Kill(int exitCode = SIGINT) {
  // Kill the process with the given code.
  kill(getpid(), exitCode);
}
```

```
TEST(MyDeathTest, Exit) {
  // Check if Exit() function exits with non-zero code.
  EXPECT_DEATH(Exit(-1), "");
}

TEST(MyDeathTest, NormalExit) {
  EXPECT_EXIT(Exit(0), testing::ExitedWithCode(0), "");
}

TEST(MyDeathTest, AbnormalExit) {
  EXPECT_EXIT(Exit(1), testing::ExitedWithCode(1), "");
}

TEST(MyDeathTest, KillProcess) {
  EXPECT_EXIT(Kill(SIGKILL),
              testing::KilledBySignal(SIGKILL), "");
}
```

Name your **test suite** (not test) **`*DeathTest`** since test suites with a name ending in "`DeathTest`" are run before all other tests.

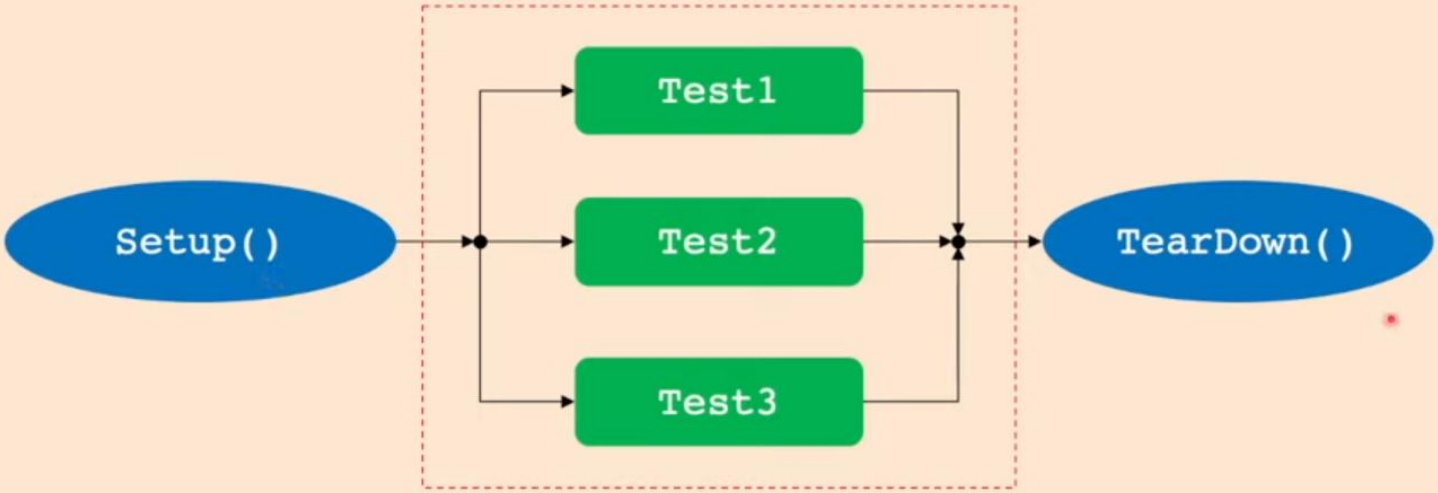
See **Death Tests And Threads** in GTest documentation

`tests/gtest_demo/death_test.cc`

Other Macros

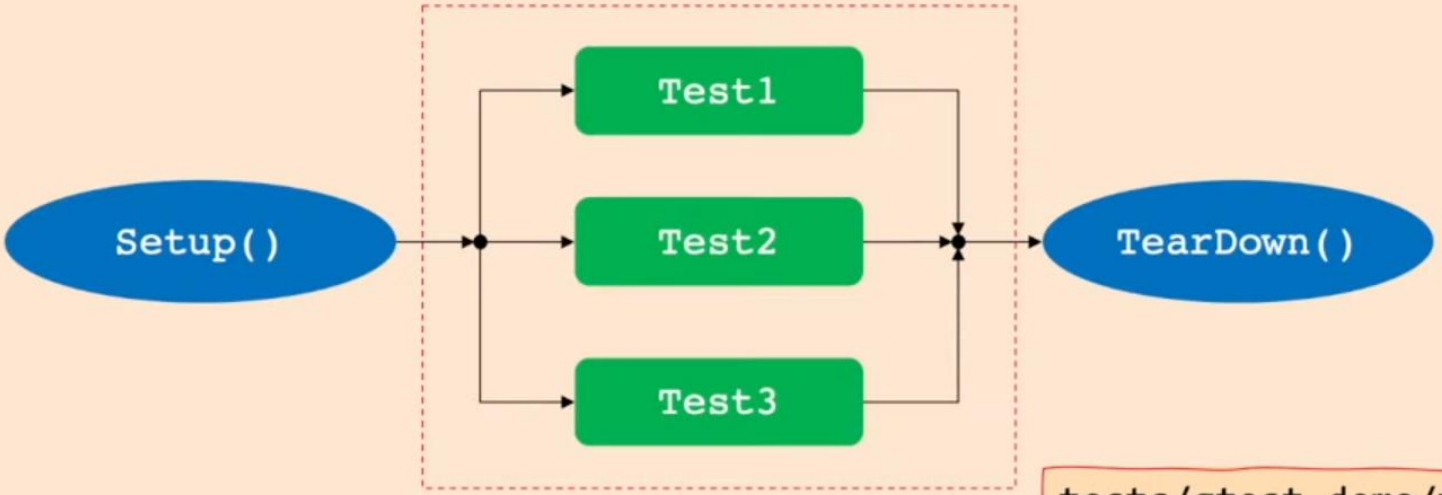
Assertion	Description
SUCCEED()	Generates a success. This *does not* make the overall test succeed. A test is considered successful only if none of its assertions fail during its execution. The SUCCEED assertion is purely documentary and currently doesn't generate any user-visible output.
FAIL()	This will generate a fatal failure.
ADD_FAILIUR()	
ADD_FAILURE_AT(*file_path*,*line_number*)	Generates a nonfatal failure at the file and line number specified.

Test Fixtures:



```
class MyTestFixture : public ::testing::Test {
  protected:

  void SetUp() override {
    ...
  }

  void TearDown() override {
    ...
  }

};
```

```
TEST_F(MyTestFixture, Test1) {
  ...
}

TEST_F(MyTestFixture, Test2) {
  ...
}

TEST_F(MyTestFixture, Test3) {
  ...
}
```



tests/gtest_demo/test_fixture.cc

Test Fixtures

```
template <typename E>
class Queue {
 private:
  std::vector<E> _v;

 public:
  Queue() {}
  void Enqueue(const E& element);
  E Dequeue;
  size_t size() const ;
  bool IsEmpty();
};
```

```
class QueueTest : public ::testing::Test {
  protected:

  void SetUp() override {
    q_.Enqueue(1);
    q_.Enqueue(2);
  }

  void TearDown() override {
    PrintQueue();
    std::cout << "=====Test ended!=====" <<
        std::endl;
  }

  void PrintQueue();

  Queue<int> q_;
};
```

```
TEST_F(QueueTest,IsNotEmptyInitially) {
  EXPECT_EQ(q_.size(), 2);
  EXPECT_FALSE(q_.IsEmpty());
}
```

Use **TEST_F** instead of TEST

tests/gtest_demo/test_fixture.cc

```cpp
class QueueTest : public ::testing::Test {
  protected:

  void SetUp() override {
    q_.Enqueue(1);
    q_.Enqueue(2);
  }

  void TearDown() override {
    PrintQueue();
    std::cout << "=====Test ended!=====" << std::endl;
  }

  void PrintQueue();

  Queue<int> q_;
};
```

```cpp
TEST_F(QueueTest, SizeAndEmptyWork) {
  EXPECT_EQ(q_.size(), 2);
  EXPECT_FALSE(q_.IsEmpty());
}

TEST_F(QueueTest, DequeueWorks) {
  EXPECT_EQ(q_.Dequeue(), 1);
  EXPECT_EQ(q_.Dequeue(), 2);
}

TEST_F(QueueTest, EnqueueWorks) {
  q_.Enqueue(3);
  EXPECT_EQ(q_.Dequeue(), 1);
  EXPECT_EQ(q_.Dequeue(), 2);
  EXPECT_EQ(q_.Dequeue(), 3);
}
```

```
exec ${PAGER:-/usr/bin/less} "$0" || exit 1
Executing tests from //tests/gtest_demo:test_fixture
-----------------------------------------------------------------------------
Running main() from gmock_main.cc
[==========] Running 5 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 5 tests from QueueTest
[ RUN      ] QueueTest.IsNotEmptyInitially
Test took: 0 ms
-> Remaining item in the queue q_.Dequeue(): 1
-> Remaining item in the queue q_.Dequeue(): 2
=========================Test ended!=========================
[       OK ] QueueTest.IsNotEmptyInitially (0 ms)
[ RUN      ] QueueTest.SizeAndEmptyWork
Test took: 0 ms
-> Remaining item in the queue q_.Dequeue(): 1
-> Remaining item in the queue q_.Dequeue(): 2
=========================Test ended!=========================
[       OK ] QueueTest.SizeAndEmptyWork (0 ms)
[ RUN      ] QueueTest.DequeueWorks
Test took: 0 ms
-> The queue is empty.
=========================Test ended!=========================
[       OK ] QueueTest.DequeueWorks (0 ms)
[ RUN      ] QueueTest.EnqueueWorks
Test took: 0 ms
-> The queue is empty.
=========================Test ended!=========================
[       OK ] QueueTest.EnqueueWorks (0 ms)
[ RUN      ] QueueTest.TakesTooLong
Test took: 1224 ms
-> Remaining item in the queue q_.Dequeue(): 1
-> Remaining item in the queue q_.Dequeue(): 1
=========================Test ended!=========================
[       OK ] QueueTest.TakesTooLong (1224 ms)
[----------] 5 tests from QueueTest (1224 ms total)

[----------] Global test environment tear-down
[==========] 5 tests from 1 test suite ran. (1224 ms total)
[  PASSED  ] 5 tests.
```

tests/gtest_demo/test_fixture.cc

Test Fixture: Set Test Timeout

```
class TestFixture : public ::testing::Test {
protected:
  void SetUp() override {
    start_time_ms_ = std::chrono::high_resolution_clock::now();
    q_.Enqueue(1);
    q_.Enqueue(2);
  }

  void TearDown() override {
    auto end_time_ms = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(
              end_time_ms - start_time_ms_);

    std::cout << "Test took: " << duration.count() << " ms" << std::endl;
    EXPECT_TRUE(duration.count() <= 1000) << "-> The test took too long!";
    PrintQueue();

    std::cout << "========================Test ended!========================"
                      << std::endl;
  }

  std::chrono::time_point<std::chrono::high_resolution_clock> start_time_ms_;
};
```

```
TEST_F(QueueTest, TakesTooLong) {
  for (unsigned long i = 0; i < 10000000; i++) {
    q_.Enqueue(1);
    q_.Dequeue();
  }
  EXPECT_EQ(q_.size(), 2);
}
```

```
[ RUN      ] QueueTest.TakesTooLong
Test took: 1226 ms
tests/gtest_demo/test_fixture.cc:47: Failure
Value of: duration.count() <= 1000
  Actual: false
Expected: true
-> The test took too long!
-> Remaining item in the queue q_.Dequeue(): 1
-> Remaining item in the queue q_.Dequeue(): 1
========================Test ended!========================
[  FAILED  ] QueueTest.TakesTooLong (1226 ms)
[----------] 5 tests from QueueTest (1226 ms total)

[----------] Global test environment tear-down
[==========] 5 tests from 1 test suite ran. (1226 ms total)
[  PASSED  ] 4 tests.
[  FAILED  ] 1 test, listed below:
[  FAILED  ] QueueTest.TakesTooLong

 1 FAILED TEST
```

tests/gtest_demo/test_fixture.cc

Test Fixtures: Sharing Resources

```
class FooTest : public testing::Test {
protected:
  static void SetUpTestSuite() {
    std::cout << "========Beginning of all tests========" << std::endl;
    if (shared_resource_ == nullptr) {
      shared_resource_ =
          new std::string("This is an expensive shared resource!");
    }
  }

  static void TearDownTestSuite() {
    std::cout << "========End of all tests========" << std::endl;
    delete shared_resource_;
    shared_resource_ = nullptr;
  }

  void SetUp() override {
    std::cout << "Called before each test." << std::endl;
  }

  void TearDown() override {
    std::cout << "Called after each test." << std::endl;
  }

  // Some expensive resource shared by all tests.
  static std::string* shared_resource_;
};
```

```
-----------------------------------------------------------------------------
Running main() from gmock_main.cc
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from FooTest
========Beginning of all tests========
[ RUN      ] FooTest.Test1
Called before each test.
Called after each test.
[       OK ] FooTest.Test1 (0 ms)
[ RUN      ] FooTest.Test2
Called before each test.
Called after each test.
[       OK ] FooTest.Test2 (0 ms)
========End of all tests========
[----------] 2 tests from FooTest (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 2 tests.
```

tests/gtest_demo/test_fixture_shared_resources.c

Manipulating Test

Disabling Tests

```
TEST(BasicChecks, DISABLED_SomeBrokenTest) {
  EXPECT_TRUE(false);
}

TEST(BasicChecks, EnabledTest) {
  EXPECT_TRUE(true);
}
```

```
INFO: Analyzed target //tests/gtest_macros:disabled_tests (1 packages loaded, 2 targets configured).
INFO: Found 1 target...
Target //tests/gtest_macros:disabled_tests up-to-date:
  bazel-bin/tests/gtest_macros/disabled_tests
INFO: Elapsed time: 0.173s, Critical Path: 0.02s
INFO: 1 process: 1 internal.
INFO: Build completed successfully, 1 total action
INFO: Build completed successfully, 1 total action
exec ${PAGER:-/usr/bin/less} "$0" || exit 1
Executing tests from //tests/gtest_macros:disabled_tests
-----------------------------------------------------------------------------
Running main() from gmock_main.cc
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from BasicChecks
[ RUN      ] BasicChecks.EnabledTest
[       OK ] BasicChecks.EnabledTest (0 ms)
[----------] 1 test from BasicChecks (0 ms total)

[----------] Global test environment tear-down
[==========] 1 test from 1 test suite ran. (0 ms total)
[  PASSED  ] 1 test.

  YOU HAVE 1 DISABLED TEST
```

```
class DISABLED_BasicChecks : public testing::Test {};

TEST_F(DISABLED_BasicChecks, SomeTest) {
 EXPECT_TRUE(false);
}
```

```
tests/gtest_macros/disabled_tests.cc
bazel test tests/gtest_macros/disabled_tests
```

Skipping Tests

```
TEST(BasicChecks, EnabledButSkippedTest) {
  GTEST_SKIP() << "Skipping single test";
  EXPECT_TRUE(false);
}
```

```
exec ${PAGER:-/usr/bin/less} "$0" || exit 1
Executing tests from //tests/gtest_macros:disabled_tests
-----------------------------------------------------------------
-------------------
Running main() from gmock_main.cc
[==========] Running 2 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 2 tests from BasicChecks
[ RUN ] BasicChecks.EnabledTest
[ OK ] BasicChecks.EnabledTest (0 ms)
[ RUN ] BasicChecks.EnabledButSkippedTest
tests/gtest_macros/disabled_tests.cc:20: Skipped
Skipping single test
[ SKIPPED ] BasicChecks.EnabledButSkippedTest (0 ms)
[----------] 2 tests from BasicChecks (0 ms total)

[----------] Global test environment tear-down
[==========] 2 tests from 1 test suite ran. (0 ms total)
[ PASSED ] 1 test.
[ SKIPPED ] 1 test, listed below:
[ SKIPPED ] BasicChecks.EnabledButSkippedTest

YOU HAVE 1 DISABLED TEST
```

```
class SomeTextFixture : public ::testing::Test {
protected:
  void SetUp() override {
    GTEST_SKIP() << "Skipping all tests for this fixture";
  }
};

// All tests in this fixture will be skipped.
TEST_F(SomeTextFixture, SomeTest1) { EXPECT_TRUE(false); }
TEST_F(SomeTextFixture, SomeTest2) { EXPECT_TRUE(false); }
```

tests/gtest_demo/disabled_tests_fixture.cc

Filtering Tests: --gtest_filter

```
TEST(BasicChecks, MyTest1) { EXPECT_TRUE(true); }

TEST(BasicChecks, MyTest2) { EXPECT_TRUE(true); }

TEST(BasicChecks, YourTest1) { EXPECT_TRUE(false); }

TEST(BasicChecks, YourTest2) { EXPECT_TRUE(false); }
```

```
bazel test tests/gtest_demo:filtered_tests --test_arg=--gtest_filter=*.My*
OR
bazel test tests/gtest_demo:filtered_tests --test_filter=*.My*
```

```
ari@Arios-MacBook-Pro:~/github/cpp-template$ bazel test tests/gtest_demo:filtered_tests --test_filter=*.My*
INFO: Analyzed target //tests/gtest_demo:filtered_tests (0 packages loaded, 0 targets configured).
INFO: Found 1 test target...
Target //tests/gtest_demo:filtered_tests up-to-date:
  bazel-bin/tests/gtest_demo/filtered_tests
INFO: Elapsed time: 1.129s, Critical Path: 1.05s
INFO: 4 processes: 1 internal, 3 darwin-sandbox.
INFO: Build completed successfully, 4 total actions
//tests/gtest_demo:filtered_tests                                      PASSED in 0.2s

Executed 1 out of 1 test: 1 test passes.
INFO: Build completed successfully, 4 total actions
ari@Arios-MacBook-Pro:~/github/cpp-template$
```

Customizing Output

Comparison Of User Types

```
struct Person {
  Person(std::string_view name, int age) :
         name(name), age(age) {}

  std::string name;
  int age;
};
```

```
TEST(PersonTest, CanAssertEqualitySimple) {
  Person p_ari("Ari", 25);
  Person p_ted("Ted", 21);

  EXPECT_EQ(p_ari.name, p_ted.name);
  EXPECT_EQ(p_ari.age, p_ted.age);
}
```

```
[ RUN      ] PersonTest.CanAssertEqualitySimple
tests/gtest_demo/user_types.cc:41: Failure
Expected equality of these values:
  p_ari.name
    Which is: "Ari"
  p_ted.name
    Which is: "Ted"
tests/gtest_demo/user_types.cc:42: Failure
Expected equality of these values:
  p_ari.age
    Which is: 25
  p_ted.age
    Which is: 21
[  FAILED  ] PersonTest.CanAssertEqualitySimple (0 ms)
```

tests/gtest_demo/user_types.cc

Printing User Types

```
struct Person {
  Person(std::string_view name, int age) :
          name(name), age(age) {}

  // Define this for EXPECT_EQ and ASSERT_EQ.
  bool operator==(const Person& rhs) const {
    return (name == rhs.name && age == rhs.age);
  }

  std::string name;
  int age;
};
```

```
TEST(PersonTest, CanDirectlyAssertEquality)
{
  Person p_ari("Ari", 25);
  Person p_ted("Ted", 21);

  EXPECT_EQ(p_ari, p_ted);
}
```

```
[ RUN      ] PersonTest.CanDirectlyAssertEquality
tests/gtest_demo/user_types.cc:50: Failure
Expected equality of these values:
  p_ari
    Which is: 32-byte object <06-41 72-69 00-00 00-00 90-40 50-30 BA-7F 00-00 90-40 50-30 BA-7F 00-00 19-00 00-00 FE-7F
    00-00>
  p_ted
    Which is: 32-byte object <06-54 65-64 00-7F 00-00 98-40 50-30 BA-7F 00-00 10-00 00-00 00-00 00-00 15-00 00-00 00-00
    00-00>
[  FAILED  ] PersonTest.CanDirectlyAssertEquality (0 ms)
```

tests/gtest_demo/user_types.cc

Printing User Types

```cpp
struct Person {
// ...
   // Google test will call either operator<< or
   // PrintTo functions to print your object.
   // If both are defined, it will use PrintTo.
  friend std::ostream& operator<<(std::ostream& os,
            const Person& person) {
    return os << person.DebugString();
  }

  friend void PrintTo(const Person& person, std::ostream* os) {
    *os << person.DebugString();
  }

  // This function will be called by testing::PrintToString.
  std::string DebugString() const {
    std::string result;
    return result + "name: " + name +
      ", age: " + std::to_string(age) + "\n";

  }
};
```

```cpp
TEST(PersonTest, CanDirectlyAssertEqualityAndPrint)
{
  Person p_ari("Ari", 25);
  Person p_ted("Ted", 21);

  EXPECT_EQ(p_ari, p_ted)
        << PrintToString(p_ari)
        << p_ted.DebugString();
}
```

```
[ RUN      ] PersonTest.CanDirectlyAssertEqualityAndPrint
tests/gtest_demo/user_types.cc:57: Failure
Expected equality of these values:
  p_ari
    Which is: name: Ari, age: 25

  p_ted
    Which is: name: Ted, age: 21

name: Ari, age: 25
name: Ted, age: 21

[  FAILED  ] PersonTest.CanDirectlyAssertEqualityAndPrint (0 ms)
```

tests/gtest_demo/user_types.cc

Sharing the Test Logic

For Various Inputs

Method 1: One Test Per Input

```
// Test with the following values
{}, // empty
{1}, // single size
{5, 3, 1, 77}, // small size random values
{5, 4, 3, 2, 1}, // reversely sorted
{-4, 122, -1000, -4, 122} // with duplicates

void SelectionSort(std::vector<int> &input) {
  for (int i = 0; i < int(input.size() - 1); i++) {
    int min_index = FindMinIndex(input, i);
    Swap(input[i], input[min_index]);
  }
}
```

```
TEST(SortTestSimple, SortsWorkForEmptyVec
  auto in = std::vector<int>{};
  auto expected = in;
  SelectionSort(in);
  std::sort(expected.begin(), expected.end());
  EXPECT_EQ(expected, in);
}

TEST(SortTestSimple, SortsWorkForSingleSizeVector) {
  auto in = std::vector<int>{1};
  auto expected = in;
  SelectionSort(in);
  std::sort(expected.begin(), expected.end());
  EXPECT_EQ(expected, in);
}

TEST(SortTestSimples, SortsWorkForReverseVector) {
  auto in = std::vector<int>{5, 4, 3, 2, 1};
  auto expected = in;
  SelectionSort(in);
  std::sort(expected.begin(), expected.end());
  EXPECT_EQ(expected, in);
}
```

Don't Repeat Yourself
Don't Repeat Yourself
Don't Repeat Yourself
Don't Repeat Yourself

👉 **Readability** of the test code is often preferred to its **conciseness**.

tests/gtest_demo/test_share_logic.cc

Method 2: Use a Test Fixture

```
TEST(SortTestSimple, SortsWorkForEmptyVector) {
  auto in = std::vector<int>{};
  auto expected = in;
  SelectionSort(in);
  std::sort(expected.begin(), expected.end());
  EXPECT_EQ(expected, in);
}

TEST(SortTestSimple, SortsWorkForSingleSizeVector) {
  auto in = std::vector<int>{1};
  auto expected = in;
  SelectionSort(in);
  std::sort(expected.begin(), expected.end());
  EXPECT_EQ(expected, in);
}

TEST(SortTestSimples, SortsWorkForReverseVector) {
  auto in = std::vector<int>{5, 4, 3, 2, 1};
  auto expected = in;
  SelectionSort(in);
  std::sort(expected.begin(), expected.end());
  EXPECT_EQ(expected, in);
}
```

```
class SortTestSimpleFixture : public ::testing::Test {
  void TearDown() override {
    expected_ = in_;
    SelectionSort(in_);
    std::sort(expected_.begin(), expected_.end());
    EXPECT_EQ(expected_, in_);
  }

public:
  std::vector<int> in_;
  std::vector<int> expected_;
};
```

```
TEST_F(SortTestSimpleFixture, SortsWorkForEmptyVector) {
  in_ = std::vector<int>{};
}

TEST_F(SortTestSimpleFixture, SortsWorkForSingleSizeVector) {
  in_ = std::vector<int>{1};
}

TEST_F(SortTestSimpleFixture, SortsWorkForReverseVector) {
  in_ = std::vector<int>{5, 4, 3, 2, 1};
}
```

tests/gtest_demo/test_share_logic.cc

Method 3: All Inputs in One Test

```
TEST(SortTestAllInputs, WorksForAllInputs) {
  std::vector<std::vector<int>> ins = {
    {}, // empty
    {1}, // single size
    {5, 3, 1, 77}, // small size random values
    {5, 4, 3, 2, 1}, // reversely sorted
    {-4, 122, -1000, -4, 122} // with duplicates
  };

  for (auto &in : ins) {
    auto expected = in;
    SelectionSort(in);
    std::sort(expected.begin(), expected.end());
    EXPECT_EQ(expected, in);
  }
}
```

```
[----------] 1 test from SortTestAllInputs
[ RUN      ] SortTestAllInputs.WorksForAllInputs
tests/gtest_demo/test_share_logic.cc:102: Failure
Expected equality of these values:
  expected
    Which is: { 5, 3, 1, 77 }
  in
    Which is: { 1, 3, 5, 77 }
tests/gtest_demo/test_share_logic.cc:102: Failure
Expected equality of these values:
  expected
    Which is: { 5, 4, 3, 2, 1 }
  in
    Which is: { 1, 2, 3, 4, 5 }
tests/gtest_demo/test_share_logic.cc:102: Failure
Expected equality of these values:
  expected
    Which is: { -4, 122, -1000, -4, 122 }
  in
    Which is: { -1000, -4, -4, 122, 122 }
[  FAILED  ] SortTestAllInputs.WorksForAllInputs (0 ms)
[----------] 1 test from SortTestAllInputs (0 ms total)
```

tests/gtest_demo/test_share_logic.cc

Method 4: Parameterized Test

```
class SortTest : public testing::TestWithParam<std::vector<int>> {};

TEST_P(SortTest, WorksForVariousInputs) {
  auto in = GetParam();
  auto expected = in;

  SelectionSort(in);
  std::sort(expected.begin(), expected.end());

  EXPECT_EQ(expected, in);
}

INSTANTIATE_TEST_SUITE_P(
          SelectionSortVariousInputs, SortTest,
          testing::Values(std::vector<int>{},
            std::vector<int>{1},
            std::vector<int>{5, 3,1 , 77},
            std::vector<int>{5, 4, 3, 2, 1},
            std::vector<int>{-4, 122, -1000, -4, 122}
));
```

```
[----------] 5 tests from SelectionSortVariousInputs/SortTest
[ RUN      ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/0
[       OK ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/0 (0 ms)
[ RUN      ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/1
[       OK ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/1 (0 ms)
[ RUN      ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/2
[       OK ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/2 (0 ms)
[ RUN      ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/3
[       OK ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/3 (0 ms)
[ RUN      ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/4
[       OK ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/4 (0 ms)
[----------] 5 tests from SelectionSortVariousInputs/SortTest (0 ms total)
```

INSTANTIATE_TEST_SUITE_P

`INSTANTIATE_TEST_SUITE_P( InstantiationName , TestSuiteName , param_generator )`
`INSTANTIATE_TEST_SUITE_P( InstantiationName , TestSuiteName , param_generator , name_generator )`

Instantiates the value-parameterized test suite `TestSuiteName` (defined with `TEST_P`).

The argument `InstantiationName` is a unique name for the instantiation of the test suite, to distinguish between multiple instantiations. In test output, the instantiation name is added as a prefix to the test suite name `TestSuiteName`.

tests/gtest_demo/test_share_logic.cc

Method 4: Parameterized Test with Name Generator

```
std::map<std::string, std::vector<int>> input_map = {
  //
  {"Empty", {}},
  {"SingleElement", {1}},
  {"ReverseSort", {5, 4, 3, 2, 1}},
  {"SmallVector", {5, 3, 1, 77}},
  {"Duplicates", {-4, 122, -1000, -4, 122, -1000}}
  //
};
class SortTestNameGenerator
  : public testing::TestWithParam<
  std::pair<const std::string, std::vector<int>>> {};

TEST_P(SortTestNameGenerator, WorksForVariousInputs) {
  auto in = GetParam().second;
  auto expected = in;
  SelectionSort(in);
  std::sort(expected.begin(), expected.end());
  EXPECT_EQ(expected, in);
}

INSTANTIATE_TEST_SUITE_P(
  SelectionSortVariousInputsNameGenerator, SortTestNameGenerator,
  testing::ValuesIn(input_map),
  [](const testing::TestParamInfo<SortTestNameGenerator::ParamType> &info) {
    return info.param.first;
  });
```

```
[----------] 5 tests from SelectionSortVariousInputsNameGenerator/SortTestNameGenerator
[ RUN      ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/Duplicates
[       OK ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/Duplicates (0 ms)
[ RUN      ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/Empty
[       OK ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/Empty (0 ms)
[ RUN      ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/ReverseSort
[       OK ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/ReverseSort (0 ms)
[ RUN      ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/SingleElement
[       OK ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/SingleElement (0 ms)
[ RUN      ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/SmallVector
[       OK ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/SmallVector (0 ms)
[----------] 5 tests from SelectionSortVariousInputsNameGenerator/SortTestNameGenerator (0 ms total)
```

tests/gtest_demo/test_share_logic.cc

```
TEST(SortTestSimple, SortsWorkForEmptyVector) {
  auto in = std::vector<int>{};
  auto expected = in;
  SelectionSort(in);
  std::sort(expected.begin(), expected.end());
  EXPECT_EQ(expected, in);
}

TEST(SortTestSimple, SortsWorkForSingleSizeVector) {
  auto in = std::vector<int>{1};
  auto expected = in;
  SelectionSort(in);
  std::sort(expected.begin(), expected.end());
  EXPECT_EQ(expected, in);
}
```

```
TEST(SortTestAllInputs, WorksForAllInputs) {
  std::vector<std::vector<int>> ins = {
    {}, // empty
    {1}, // single size
    {5, 3, 1, 77}, // small size random values
    {5, 4, 3, 2, 1}, // reversely sorted
    {-4, 122, -1000, -4, 122} // with duplicates
  };

  for (auto &in : ins) {
    auto expected = in;
    SelectionSort(in);
    std::sort(expected.begin(), expected.end());
    EXPECT_EQ(expected, in);
  }
}
```

```
class SortTest : public testing::TestWithParam<std::vector<int>> {};

TEST_P(SortTest, WorksForVariousInputs) {
  auto in = GetParam();
  auto expected = in;
  SelectionSort(in);
  std::sort(expected.begin(), expected.end());
  EXPECT_EQ(expected, in);
}

INSTANTIATE_TEST_SUITE_P(
            SelectionSortVariousInputs, SortTest,
            testing::Values(std::vector<int>{},
              std::vector<int>{1},
              std::vector<int>{5, 3,1 , 77},
              std::vector<int>{5, 4, 3, 2, 1},
              std::vector<int>{-4, 122, -1000, -4, 122}
));
```

```
std::map<std::string, std::vector<int>> input_map = {
  //
  {"Empty", {}},
  {"SingleElement", {1}},
  {"ReverseSort", {5, 4, 3, 2, 1}},
  {"SmallVector", {5, 3, 1, 77}},
  {"Duplicates", {-4, 122, -1000, -4, 122, -1000}}
  //
};
class SortTestNameGenerator
  : public testing::TestWithParam<
  std::pair<const std::string, std::vector<int>>> {};

TEST_P(SortTestNameGenerator, WorksForVariousInputs) {
  auto in = GetParam().second;
  auto expected = in;
  SelectionSort(in);
  std::sort(expected.begin(), expected.end());
  EXPECT_EQ(expected, in);
}

INSTANTIATE_TEST_SUITE_P(
  SelectionSortVariousInputsNameGenerator, SortTestNameGenerator,
  testing::ValuesIn(input_map),
  [](const testing::TestParamInfo<
            SortTestNameGenerator::ParamType> &info) {
    return info.param.first;
  });
```

```
[----------] 5 tests from SelectionSortVariousInputs/SortTest
[ RUN      ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/0
[       OK ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/0 (0 ms)
[ RUN      ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/1
[       OK ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/1 (0 ms)
[ RUN      ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/2
[       OK ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/2 (0 ms)
[ RUN      ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/3
[       OK ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/3 (0 ms)
[ RUN      ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/4
[       OK ] SelectionSortVariousInputs/SortTest.WorksForVariousInputs/4 (0 ms)
[----------] 5 tests from SelectionSortVariousInputs/SortTest (0 ms total)
```

```
[----------] 5 tests from SelectionSortVariousInputsNameGenerator/SortTestNameGenerator
[ RUN      ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/Empty
[       OK ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/Empty (0 ms)
[ RUN      ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/SingleElement
[       OK ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/SingleElement (0 ms)
[ RUN      ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/SmallVector
[       OK ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/SmallVector (0 ms)
[ RUN      ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/ReverseSort
[       OK ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/ReverseSort (0 ms)
[ RUN      ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/Duplicates
[       OK ] SelectionSortVariousInputsNameGenerator/SortTestNameGenerator.WorksForVariousInputs/Duplicates (0 ms)
[----------] 5 tests from SelectionSortVariousInputsNameGenerator/SortTestNameGenerator (0 ms total)
```

Sharing the Test Logic and Inputs

For Various Implementations

Testing Multiple Implementations
Method 1: Function Pointer

```
void SelectionSort(std::vector<int> &input) {
  for (int i = 0; i < int(input.size() - 1); i++) {
    int min_index = FindMinIndex(input, i);
    Swap(input[i], input[min_index]);
  }
}

void BubbleSort(std::vector<int> &input) {
  bool go;
  do {
    go = false;
    for (int i = 0; i < int(input.size() - 1); i++) {
      if (input[i] > input[i + 1]) {
       Swap(input[i], input[i + 1]);
       go = true;
      }
    }
  } while (go);
}
```

```
template <class T>
void TestSort(T sort_func) {
  std::vector<int> in;
  std::vector<int> expected;

  // Case 1: Check small vector
  in = {5, 3, 1, 77, -1};
  sort_func(in);
  expected = {-1, 1, 3, 5, 77};
  EXPECT_EQ(expected, in);
  EXPECT_EQ(expected.size(), in.size());

  // Case 2: ...
  // Case 3: ...
  // ...
}

TEST(SortTest, SelectionSort) {
 TestSort(SelectionSort);
}
TEST(SortTest, BubbleSort) {
  TestSort(BubbleSort);
}
```

tests/gtest_demo/test_function.cc

Testing Multiple Implementations
Method 2: Everything in One Test

```
TEST(SortTestAllAtOnce, WorksOnAllFunctionsWithAllInputs) {
  std::vector<std::vector<int>> ins = {
    {}, // empty
    {1}, // single size
    {5, 3, 1, 77}, // small size random values
    {5, 4, 3, 2, 1}, // reversely sorted
    {-4, 122, -1000, -4, 122} // with duplicates
  };

  std::vector<std::function<void(std::vector<int> &)>>
          functions = {
    SelectionSort, BubbleSort
  };

  // Applying all inputs and all functions at once!
  for (auto f : functions) {
    for (auto &in : ins) {
      auto expected = in;
      f(in);
      std::sort(expected.begin(), expected.end());
      EXPECT_EQ(expected, in);
    }
  }
}
```

```
[ RUN      ] SortTestAllAtOnce.WorksOnAllFunctionsWithAllInputs
tests/gtest_demo/test_function.cc:187: Failure
Expected equality of these values:
  expected
    Which is: { 5, 3, 1, 77 }
  in
    Which is: { 1, 3, 5, 77 }
tests/gtest_demo/test_function.cc:187: Failure
Expected equality of these values:
  expected
    Which is: { 5, 4, 3, 2, 1 }
  in
    Which is: { 1, 2, 3, 4, 5 }
tests/gtest_demo/test_function.cc:187: Failure
Expected equality of these values:
  expected
    Which is: { -4, 122, -1000, -4, 122 }
  in
    Which is: { -1000, -4, -4, 122, 122 }
[  FAILED  ] SortTestAllAtOnce.WorksOnAllFunctionsWithAllInputs (0 ms)
[----------] 1 test from SortTestAllAtOnce (0 ms total)
```

tests/gtest_demo/test_function.cc

Testing Multiple Implementations
Method 3: Parameterized Test

```
class SortTest
: public testing::TestWithParam<std::function<void(std::vector<int> &)> > {};

INSTANTIATE_TEST_SUITE_P(SelectionSortBubbleSort, SortTest,
          testing::Values(SelectionSort, BubbleSort));
```

```
TEST_P(SortTest, EmptyCollection) {
  std::vector<int> in = {};
  std::vector<int> expected = {};
  auto sort_func = GetParam();

  sort_func(in);
  EXPECT_EQ(expected, in);
  EXPECT_EQ(expected.size(), in.size());
}
```

```
TEST_P(SortTest, SmallNumberOfItems) {
  std::vector<int> in = {1, 11, 2, 5, 12, 9, 4, 10};
  std::vector<int> expected = {1, 2, 4, 5, 9, 10, 11, 12};
  auto sort_func = GetParam();

  sort_func(in);
  EXPECT_EQ(expected, in);
  EXPECT_EQ(expected.size(), in.size());
}
```

INSTANTIATE_TEST_SUITE_P

`INSTANTIATE_TEST_SUITE_P( InstantiationName , TestSuiteName , param_generator )`
`INSTANTIATE_TEST_SUITE_P( InstantiationName , TestSuiteName , param_generator , name_generator )`

Instantiates the value-parameterized test suite `TestSuiteName` (defined with `TEST_P`).

The argument `InstantiationName` is a unique name for the instantiation of the test suite, to distinguish between multiple instantiations. In test output, the instantiation name is added as a prefix to the test suite name `TestSuiteName`.

tests/gtest_demo/test_param.cc

Testing Multiple Implementations
Method 3: Parameterized Test

```
class SortTest
: public
testing::TestWithParam<std::function<void(std::vector<int> &)>
> {};

INSTANTIATE_TEST_SUITE_P(SelectionSortBubbleSort, SortTest,
          testing::Values(SelectionSort, BubbleSort));
```

```
Running main() from gmock_main.cc
[==========] Running 18 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 18 tests from SelectionSortBubbleSort/SortTest
[ RUN      ] SelectionSortBubbleSort/SortTest.SmallNumberOfItems/0
[       OK ] SelectionSortBubbleSort/SortTest.SmallNumberOfItems/0 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.SmallNumberOfItems/1
[       OK ] SelectionSortBubbleSort/SortTest.SmallNumberOfItems/1 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.EmptyCollection/0
[       OK ] SelectionSortBubbleSort/SortTest.EmptyCollection/0 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.EmptyCollection/1
[       OK ] SelectionSortBubbleSort/SortTest.EmptyCollection/1 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.SingleElement/0
[       OK ] SelectionSortBubbleSort/SortTest.SingleElement/0 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.SingleElement/1
[       OK ] SelectionSortBubbleSort/SortTest.SingleElement/1 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.ReverseOrder/0
[       OK ] SelectionSortBubbleSort/SortTest.ReverseOrder/0 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.ReverseOrder/1
[       OK ] SelectionSortBubbleSort/SortTest.ReverseOrder/1 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WithDuplicates/0
[       OK ] SelectionSortBubbleSort/SortTest.WithDuplicates/0 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WithDuplicates/1
[       OK ] SelectionSortBubbleSort/SortTest.WithDuplicates/1 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.RandomLargeVector/0
[       OK ] SelectionSortBubbleSort/SortTest.RandomLargeVector/0 (100 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.RandomLargeVector/1
[       OK ] SelectionSortBubbleSort/SortTest.RandomLargeVector/1 (401 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.RandomLargeVectorSorted/0
[       OK ] SelectionSortBubbleSort/SortTest.RandomLargeVectorSorted/0 (93 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.RandomLargeVectorSorted/1
[       OK ] SelectionSortBubbleSort/SortTest.RandomLargeVectorSorted/1 (3 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.RandomLargeVectorReverseSorted/0
[       OK ] SelectionSortBubbleSort/SortTest.RandomLargeVectorReverseSorted/0 (121 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.RandomLargeVectorReverseSorted/1
[       OK ] SelectionSortBubbleSort/SortTest.RandomLargeVectorReverseSorted/1 (412 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.RandomLargeVectorMultipleSeeds/0
[       OK ] SelectionSortBubbleSort/SortTest.RandomLargeVectorMultipleSeeds/0 (228 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.RandomLargeVectorMultipleSeeds/1
[       OK ] SelectionSortBubbleSort/SortTest.RandomLargeVectorMultipleSeeds/1 (1062 ms)
[----------] 18 tests from SelectionSortBubbleSort/SortTest (2424 ms total)
```

tests/gtest_demo/test_param.cc

Combining Parameters

```
TEST_P(SortTest, WorksForVariousInputs) {

  auto in = std::get<1>(GetParam());
  auto SortFunction = std::get<0>(GetParam());

  auto expected = in;
  std::sort(expected.begin(), expected.end());

  SortFunction(in);
  EXPECT_EQ(expected, in);
}
```

```
class SortTest
: public testing::TestWithParam<std::function<void(std::vector<int> &)> > {};

INSTANTIATE_TEST_SUITE_P(SelectionSortBubbleSort, SortTest,
         testing::Values(SelectionSort, BubbleSort));
```

Parameter Generator	Behavior
`Range(begin, end [, step])`	Yields values `{begin, begin+step, begin+step+step, ...}`. The values do not include `end`. `step` defaults to 1.
`Values(v1, v2, ..., vN)`	Yields values `{v1, v2, ..., vN}`.
`ValuesIn(container)` or `ValuesIn(begin,end)`	Yields values from a C-style array, an STL-style container, or an iterator range `[begin, end)`.
`Bool()`	Yields sequence `{false, true}`.
`Combine(g1, g2, ..., gN)`	Yields as `std::tuple` *n*-tuples all combinations (Cartesian product) of the values generated by the given *n* generators `g1`, `g2`, ..., `gN`.

```
class SortTest
: public testing::TestWithParam<
            std::tuple<
                    std::function<void(std::vector<int> &)>,
                    std::vector<int>
            >
          > {};

INSTANTIATE_TEST_SUITE_P(
  SelectionSortBubbleSort, SortTest,

  Combine(Values(SelectionSort, BubbleSort),
          Values(   std::vector<int>{},
                    std::vector<int>{1},
                    std::vector<int>{5, 3, 1, 77},
                    std::vector<int>{5, 4, 3, 2, 1},
                    std::vector<int>{-4, 122, -1000, -4, 122, -1000}
)));
```

```
TEST_P(SortTest, WorksForVariousInputs) {

  auto in = std::get<1>(GetParam());
  auto SortFunction = std::get<0>(GetParam());

  auto expected = in;
  std::sort(expected.begin(), expected.end());

  SortFunction(in);
  EXPECT_EQ(expected, in);
}
```

```
Running main() from gmock_main.cc
[==========] Running 10 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 10 tests from SelectionSortBubbleSort/SortTest
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/0
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/0 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/1
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/1 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/2
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/2 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/3
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/3 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/4
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/4 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/5
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/5 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/6
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/6 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/7
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/7 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/8
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/8 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/9
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/9 (0 ms)
[----------] 10 tests from SelectionSortBubbleSort/SortTest (0 ms total)

[----------] Global test environment tear-down
[==========] 10 tests from 1 test suite ran. (0 ms total)
[  PASSED  ] 10 tests.
```

sort_test_function_test_param_share_logic

```
std::map<std::string, std::vector<int>> input_map = {
  {"Empty", {}},
  {"SingleElement", {1}},
  {"ReverseSort", {5, 4, 3, 2, 1}},
  {"SmallVector", {5, 3, 1, 77}},
  {"Duplicates", {-4, 122, -1000, -4, 122, -1000}}
};

std::map<std::string,
        std::function<void(std::vector<int> &)>> function_map = {
  {"BubbleSort", BubbleSort},
  {"SelectionSort", SelectionSort}
};
```

```
class SortTest
: public testing::TestWithParam<std::tuple<
 std::pair<const std::string, std::function<void(std::vector<int> &)>>,
 std::pair<const std::string, std::vector<int>>>> {};

INSTANTIATE_TEST_SUITE_P(
  SelectionSortBubbleSortCustomNames, SortTest,
  Combine(ValuesIn(function_map), ValuesIn(input_map)),
  [](const testing::TestParamInfo<SortTest::ParamType> &info) {
    return std::get<0>(info.param).first + "_" +
           std::get<1>(info.param).first;
  });

TEST_P(SortTest, WorksForVariousInputs) {
  auto p = GetParam();
  auto SortFunction = std::get<0>(p).second;
  auto in = std::get<1>(p).second;
  auto expected = in;
  SortFunction(in);
  std::sort(expected.begin(), expected.end());
  EXPECT_EQ(in, expected);
}
```

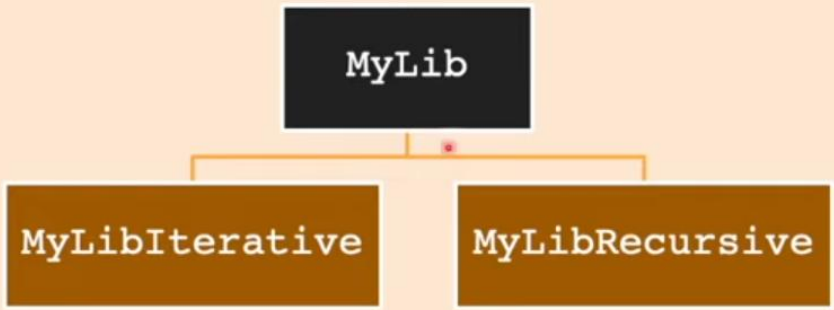
```
[----------] 10 tests from SelectionSortBubbleSort/SortTest
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/0
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/0 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/1
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/1 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/2
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/2 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/3
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/3 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/4
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/4 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/5
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/5 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/6
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/6 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/7
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/7 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/8
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/8 (0 ms)
[ RUN      ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/9
[       OK ] SelectionSortBubbleSort/SortTest.WorksForVariousInputs/9 (0 ms)
```

```
Running main() from gmock_main.cc
[==========] Running 10 tests from 1 test suite.
[----------] Global test environment set-up.
[----------] 10 tests from SelectionSortBubbleSortCustomNames/SortTest
[ RUN      ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/BubbleSort_Duplicates
[       OK ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/BubbleSort_Duplicates (0 ms)
[ RUN      ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/BubbleSort_Empty
[       OK ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/BubbleSort_Empty (0 ms)
[ RUN      ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/BubbleSort_ReverseSort
[       OK ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/BubbleSort_ReverseSort (0 ms)
[ RUN      ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/BubbleSort_SingleElement
[       OK ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/BubbleSort_SingleElement (0 ms)
[ RUN      ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/BubbleSort_SmallVector
[       OK ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/BubbleSort_SmallVector (0 ms)
[ RUN      ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/SelectionSort_Duplicates
[       OK ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/SelectionSort_Duplicates (0 ms)
[ RUN      ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/SelectionSort_Empty
[       OK ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/SelectionSort_Empty (0 ms)
[ RUN      ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/SelectionSort_ReverseSort
[       OK ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/SelectionSort_ReverseSort (0 ms)
[ RUN      ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/SelectionSort_SingleElement
[       OK ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/SelectionSort_SingleElement (0 ms)
[ RUN      ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/SelectionSort_SmallVector
[       OK ] SelectionSortBubbleSortCustomNames/SortTest.WorksForVariousInputs/SelectionSort_SmallVector (0 ms)
[----------] 10 tests from SelectionSortBubbleSortCustomNames/SortTest (0 ms total)
```

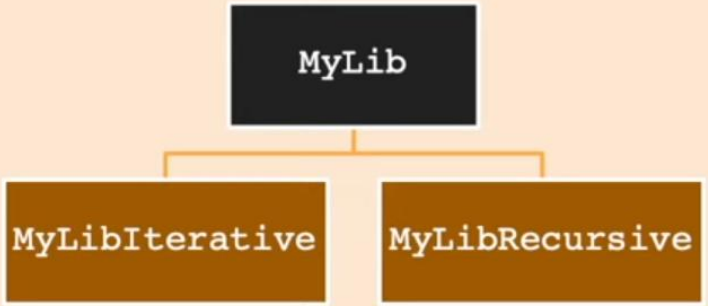
tests/gtest_demo/test_param_share_logic_name_generator.cc

Typed Tests

- **Parameterized** Tests: **Sharing** the Test **Logic** using **Parameters**
- Testing **implementations** of an **interface**
  - **Typed Test**
  - **Type-Parameterized** Test



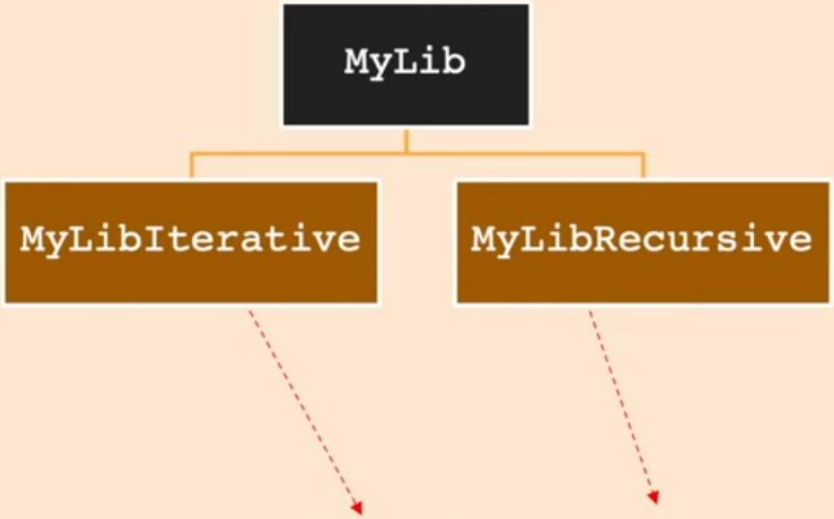
```
class MyLib {
public:
  virtual ~MyLib() {}
  // Finds the max element in a vector
  virtual int FindMax(const std::vector<int>& inputs) const = 0;
};
```




```
class MyLibIterative : public MyLib {
public:
  // Finds the max element in a vector
  virtual int FindMax(const std::vector<int>& inputs) const override {
    if (inputs.size() == 0) {
      return -1;
    }

    int result = std::numeric_limits<int>::min();
    for (auto n : inputs) {
      if (n > result) {
        result = n;
      }
    }
    return result;
  }
};
```

```
class MyLibRecursive : public MyLib {
public:
// Finds the max element in a vector

  virtual int FindMax(const std::vector<int>& inputs) const override {
    if (inputs.size() == 0) {
      return -1;
    }
    return FindMaxRecursiveAux(inputs, 0, inputs.size() - 1);
  }

private:
   int FindMaxRecursiveAux(const std::vector<int>& inputs, int left, int right) const {
    if (left >= right) {
      return inputs[left];
    }
    int mid = (right + left) / 2;
    return std::max(FindMaxRecursiveAux(inputs, left, mid),
                    FindMaxRecursiveAux(inputs, mid + 1, right));
  }
};
```

Single Test Multiple Implementations

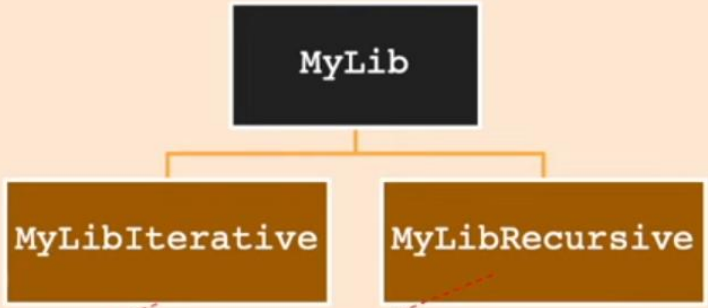



```
class MyLib {
public:
  virtual ~MyLib() {}
  // Finds the max element in a vector
  virtual int FindMax(const std::vector<int>& inputs) const = 0;
};
```

```
class MyLibIterative : public MyLib {
  virtual int FindMax(const std::vector<int>& inputs) const;
};

class MyLibRecursive : public MyLib {
  virtual int FindMax(const std::vector<int>& inputs) const;
};
```

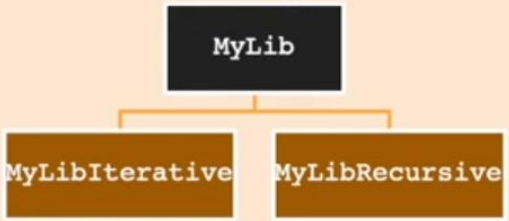
```
TYPED_TEST(MyLibTest, FindMaxHandlesNonConsecutiveNumbers) {
  std::vector<int> inputs = {1, 12, 8, 22, 2, 18, 3, 1, 4};
  EXPECT_EQ(this->lib_.FindMax(inputs), 22);
}
```

```
template <class T>
class MyLibTest : public testing::Test {
  public:
    T lib_;
};
```




```
using testing::Types;
// The list of types we want to test.
typedef Types<MyLibIterative, MyLibRecursive> Implementations;
TYPED_TEST_SUITE(MyLibTest, Implementations);
```

```
TYPED_TEST(MyLibTest, FindMaxHandlesEmpty) {
  std::vector<int> inputs = {};
  EXPECT_EQ(this->lib_.FindMax(inputs), -1);
}

TYPED_TEST(MyLibTest, FindMaxHandlesPositiveSizeOne) {
  std::vector<int> inputs = {2};
  EXPECT_EQ(this->lib_.FindMax(inputs), 2);
}

TYPED_TEST(MyLibTest, FindMaxHandlesConsecutiveNumbers) {
  std::vector<int> inputs = {1, 2, 3, 4};
  EXPECT_EQ(this->lib_.FindMax(inputs), 4);
}
```




```
// The list of types we want to test.
typedef Types<MyLibIterative, MyLibRecursive> Implementations;
TYPED_TEST_SUITE(MyLibTest, Implementations);
```

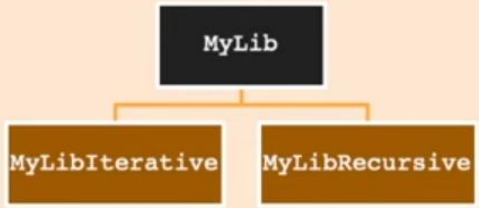
```
TYPED_TEST(MyLibTest, FindMaxHandlesEmpty) {
  std::vector<int> inputs = {};
  EXPECT_EQ(this->lib_.FindMax(inputs), -1);
}

TYPED_TEST(MyLibTest, FindMaxHandlesPositiveSizeOne) {
  std::vector<int> inputs = {2};
  EXPECT_EQ(this->lib_.FindMax(inputs), 2);
}

TYPED_TEST(MyLibTest, FindMaxHandlesConsecutiveNumbers) {
  std::vector<int> inputs = {1, 2, 3, 4};
  EXPECT_EQ(this->lib_.FindMax(inputs), 4);
}
```

```
exec ${PAGER:-/usr/bin/less} "$0" || exit 1
Executing tests from //tests/gtest_macros:typed_tests_findmax
-----------------------------------------------------------------------------
Running main() from gmock_main.cc
[==========] Running 10 tests from 2 test suites.
[----------] Global test environment set-up.
[----------] 5 tests from MyLibTest/0, where TypeParam = MyLibIterative
[ RUN      ] MyLibTest/0.FindMaxHandlesEmpty
[       OK ] MyLibTest/0.FindMaxHandlesEmpty (0 ms)
[ RUN      ] MyLibTest/0.FindMaxHandlesPositiveSizeOne
[       OK ] MyLibTest/0.FindMaxHandlesPositiveSizeOne (0 ms)
[ RUN      ] MyLibTest/0.FindMaxHandlesConsecutiveNumbers
[       OK ] MyLibTest/0.FindMaxHandlesConsecutiveNumbers (0 ms)
[ RUN      ] MyLibTest/0.FindMaxHandlesNonConsecutiveNumbers
[       OK ] MyLibTest/0.FindMaxHandlesNonConsecutiveNumbers (0 ms)
[ RUN      ] MyLibTest/0.FindMaxHandlesAllDuplicateNumbers
[       OK ] MyLibTest/0.FindMaxHandlesAllDuplicateNumbers (0 ms)
[----------] 5 tests from MyLibTest/0 (0 ms total)

[----------] 5 tests from MyLibTest/1, where TypeParam = MyLibRecursive
[ RUN      ] MyLibTest/1.FindMaxHandlesEmpty
[       OK ] MyLibTest/1.FindMaxHandlesEmpty (0 ms)
[ RUN      ] MyLibTest/1.FindMaxHandlesPositiveSizeOne
[       OK ] MyLibTest/1.FindMaxHandlesPositiveSizeOne (0 ms)
[ RUN      ] MyLibTest/1.FindMaxHandlesConsecutiveNumbers
[       OK ] MyLibTest/1.FindMaxHandlesConsecutiveNumbers (0 ms)
[ RUN      ] MyLibTest/1.FindMaxHandlesNonConsecutiveNumbers
[       OK ] MyLibTest/1.FindMaxHandlesNonConsecutiveNumbers (0 ms)
[ RUN      ] MyLibTest/1.FindMaxHandlesAllDuplicateNumbers
[       OK ] MyLibTest/1.FindMaxHandlesAllDuplicateNumbers (0 ms)
[----------] 5 tests from MyLibTest/1 (0 ms total)

[----------] Global test environment tear-down
[==========] 10 tests from 2 test suites ran. (0 ms total)
[  PASSED  ] 10 tests.
```

tests/gtest_demo/typed_tests_findmax.cc

Type-Parameterized Tests

- **Parameterized** Tests: **Sharing** the Test **Logic** using **Parameters**
- Testing **implementations** of an **interface**
  - **Typed Test**
  - **Type-Parameterized** Test




```
TYPED_TEST(MyLibTest, FindMaxHandlesEmpty) {
  std::vector<int> inputs = {};
  EXPECT_EQ(this->lib_.FindMax(inputs), -1);
}

TYPED_TEST(MyLibTest, FindMaxHandlesPositiveSizeOne) {
  std::vector<int> inputs = {2};
  EXPECT_EQ(this->lib_.FindMax(inputs), 2);
}

TYPED_TEST(MyLibTest, FindMaxHandlesConsecutiveNumbers) {
  std::vector<int> inputs = {1, 2, 3, 4};
  EXPECT_EQ(this->lib_.FindMax(inputs), 4);
}
```

```
// The list of types we want to test.
typedef Types<MyLibIterative, MyLibRecursive> Implementations;
TYPED_TEST_SUITE(MyLibTest, Implementations);
```

This does not compile if **this** is not defined first.

tests/gtest_demo/typed_tests_findmax.cc

Type-Parameterized Test

```
TYPED_TEST_SUITE_P(MyLibTest);

TYPED_TEST_P(MyLibTest, FindMaxHandlesEmpty) {
  std::vector<int> inputs = {};
  EXPECT_EQ(this->lib_.FindMax(inputs), -1);
}

TYPED_TEST_P(MyLibTest, FindMaxHandlesPositiveSizeOne) {
  std::vector<int> inputs = {2};
  EXPECT_EQ(this->lib_.FindMax(inputs), 2);
}

TYPED_TEST_P(MyLibTest, FindMaxHandlesConsecutiveNumbers) {
  std::vector<int> inputs = {1, 2, 3, 4};
  EXPECT_EQ(this->lib_.FindMax(inputs), 4);
}

REGISTER_TYPED_TEST_SUITE_P(MyLibTest,
                            FindMaxHandlesEmpty,
                            FindMaxHandlesPositiveSizeOne,
                            FindMaxHandlesConsecutiveNumbers,
                            FindMaxHandlesNonConsecutiveNumbers,
                            FindMaxHandlesAllDuplicateNumbers);
```

This can come after **this.**

```
typedef Types<MyLibIterative, MyLibRecursive> Implementations;

INSTANTIATE_TYPED_TEST_SUITE_P(IterativeAndRecursive,
                               MyLibTest,
                               Implementations);
```

Typed Test

```
typedef Types<MyLibIterative, MyLibRecursive> Implementations;
TYPED_TEST_SUITE(MyLibTest, Implementations);
```

This must come before **this.**

```
TYPED_TEST(MyLibTest, FindMaxHandlesEmpty) {
  std::vector<int> inputs = {};
  EXPECT_EQ(this->lib_.FindMax(inputs), -1);
}

TYPED_TEST(MyLibTest, FindMaxHandlesPositiveSizeOne) {
  std::vector<int> inputs = {2};
  EXPECT_EQ(this->lib_.FindMax(inputs), 2);
}

TYPED_TEST(MyLibTest, FindMaxHandlesConsecutiveNumbers) {
  std::vector<int> inputs = {1, 2, 3, 4};
  EXPECT_EQ(this->lib_.FindMax(inputs), 4);
}
```

tests/gtest_demo/type_prameterized_tests_findmax.

Testing Privat Code

```
// foo.h
class Foo {
public:
  int PublicMethod(void* x);

private:
  int PrivateMethod(void* x);
};
```

Do you really need to test this?

```
// foo.h
class Foo {
public:
  int PublicMethod(void* x);

private:
  FRIEND_TEST(FooTest, PrivateMethod);
  int PrivateMethod(void* x);
};
```

👉Try to only test the public interface, not private methods.

```
TEST(FooTest, PrivateMethod) {
 Foo foo;
 // Uses Foo's private member.
 EXPECT_EQ(foo.PrivateMethod(nullptr), 0);
}
```

Asserting in Subroutines

```
9   void MyFunction(int n) {
10    // Checks if n is 2.
11    EXPECT_EQ(n, 2);
12  }
13
14  TEST(SubTest, WithoutScopeTraceSimple) {
15    MyFunction(1);
16    MyFunction(1);
17    MyFunction(1);
18  }
```

```
8   [ RUN      ] SubTest.WithoutScopeTraceSimple
9   tests/gtest_demo/adding_traces.cc:11: Failure
10  Expected equality of these values:
11    n
12      Which is: 1
13    2
14  tests/gtest_demo/adding_traces.cc:11: Failure
15  Expected equality of these values:
16    n
17      Which is: 1
18    2
19  tests/gtest_demo/adding_traces.cc:11: Failure
20  Expected equality of these values:
21    n
22      Which is: 1
23    2
24  [  FAILED  ] SubTest.WithoutScopeTraceSimple (0 ms)
25  [ RUN      ] SubTest.WithoutScopeTraceNestedScopes
26  tests/gtest_demo/adding_traces.cc:11: Failure
```

```
32  TEST(SubTest, WithScopeTrace) {
33    {
34      SCOPED_TRACE("MyScope1");
35
36      MyFunction(1);
37      {
38        SCOPED_TRACE("MyScope2");
39        MyFunction(1);
40      }
41    }
42    MyFunction(1);
43  }
```

```
[ RUN      ] SubTest.WithScopeTrace
tests/gtest_demo/adding_traces.cc:10: Failure
Expected equality of these values:
  n
    Which is: 1
  2
Google Test trace:
tests/gtest_demo/adding_traces.cc:34: MyScope1
---------------------------------------
tests/gtest_demo/adding_traces.cc:10: Failure
Expected equality of these values:
  n
    Which is: 1
  2
Google Test trace:
tests/gtest_demo/adding_traces.cc:38: MyScope2
tests/gtest_demo/adding_traces.cc:34: MyScope1
---------------------------------------
tests/gtest_demo/adding_traces.cc:10: Failure
Expected equality of these values:
  n
    Which is: 1
  2
---------------------------------------
[  FAILED  ] SubTest.WithScopeTrace (0 ms)
```

tests/gtest_demo/adding_traces.cc

Command Line Arguments

Command Line Arguments

```
Test Selection:
--gtest_list_tests
List the names of all tests instead of running them. The name of
TEST(Foo, Bar) is "Foo.Bar".
--gtest_filter=POSITIVE_PATTERNS[-NEGATIVE_PATTERNS]
Run only the tests whose name matches one of the positive patterns but
none of the negative patterns. '?' matches any single character; '*'
matches any substring; ':' separates two patterns.
--gtest_also_run_disabled_tests
Run all disabled tests too.

Test Execution:
--gtest_repeat=[COUNT]
Run the tests repeatedly; use a negative count to repeat forever.
--gtest_shuffle
Randomize tests' orders on every iteration.
--gtest_random_seed=[NUMBER]
Random number seed to use for shuffling test orders (between 1 and
99999, or 0 to use a seed based on the current time).
```

```
Test Output:
--gtest_color=(yes|no|auto)
Enable/disable colored output. The default is auto.
--gtest_brief=1
Only print test failures.
--gtest_print_time=0
Don't print the elapsed time of each test.
--gtest_output=(json|xml)[:DIRECTORY_PATH/|:FILE_PATH]
Generate a JSON or XML report in the given directory or with the given
file name. FILE_PATH defaults to test_detail.xml.

Assertion Behavior:
--gtest_death_test_style=(fast|threadsafe)
Set the default death test style.
--gtest_break_on_failure
Turn assertion failures into debugger break-points.
--gtest_throw_on_failure
Turn assertion failures into C++ exceptions for use by an external
test framework.
--gtest_catch_exceptions=0
Do not report exceptions as test failures. Instead, allow them
to crash the program or throw a pop-up (on Windows).
```

```
bazel test tests/gtest_macros/disabled_tests --test_arg="--help" --test_output=all
```

Except for --gtest_list_tests, you can alternatively set the corresponding environment variable of a flag (all letters in upper-case).
Example:
--gtest_color=no -> GTEST_COLOR=no