Advanced C++

Alexandre Bergel
Pablo Pizarro
DCC - University of Chile
http://bergel.eu

03-01-2022

Roadmap

1. Tests (fixture, mock)
2. Debugging

Roadmap

1. **Tests (fixture, mock)**
2. Debugging

Googletest

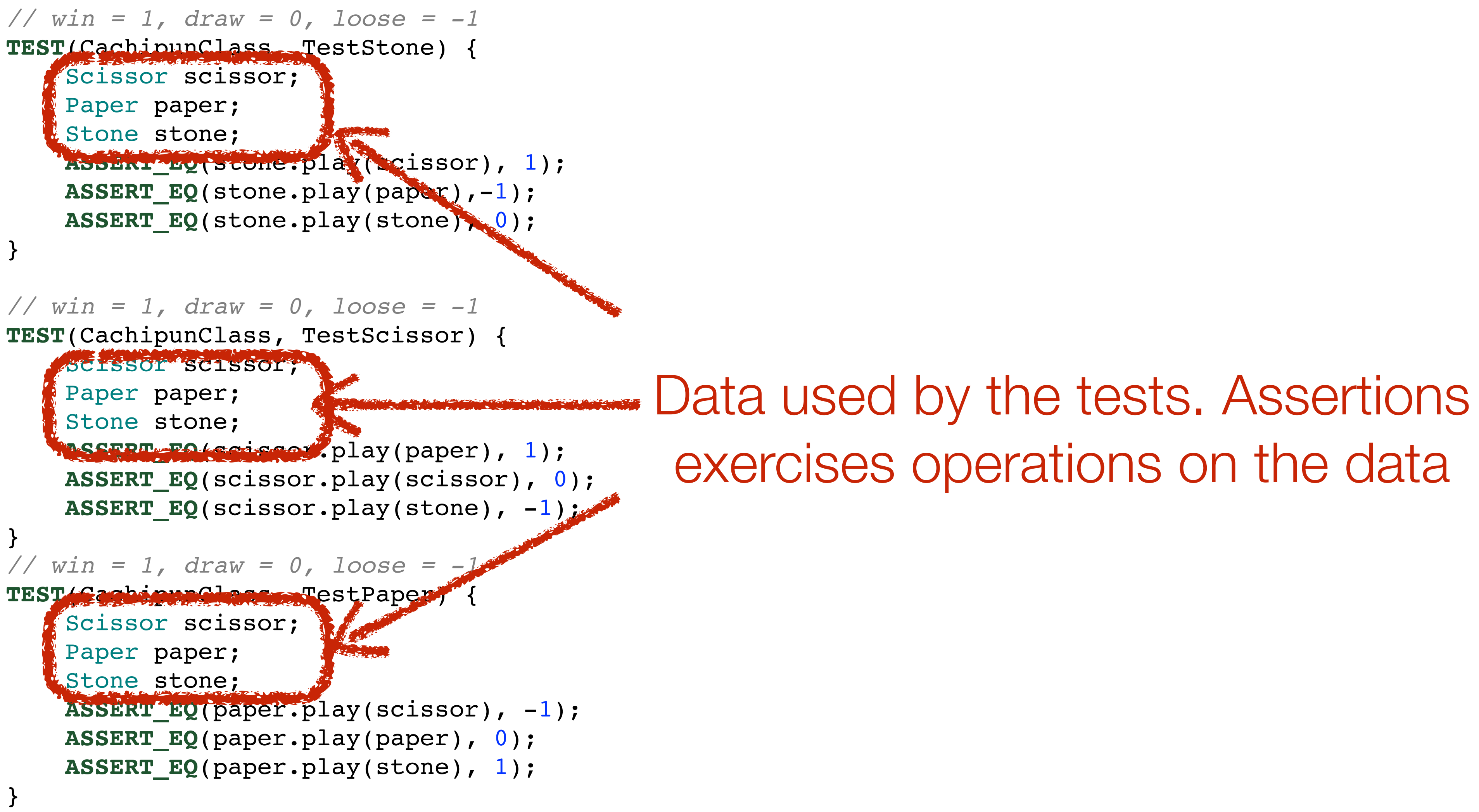
Popular framework to write unit tests in C++

Allow one to define *assertions*, *unit tests*, *test suite*

Googletest - Example

```
// win = 1, draw = 0, loose = -1
TEST(CachipunClass, TestStone) {
    Scissor scissor;
    Paper paper;
    Stone stone;
    ASSERT_EQ(stone.play(scissor), 1);
    ASSERT_EQ(stone.play(paper),-1);
    ASSERT_EQ(stone.play(stone), 0);
}

// win = 1, draw = 0, loose = -1
TEST(CachipunClass, TestScissor) {
    Scissor scissor;
    Paper paper;
    Stone stone;
    ASSERT_EQ(scissor.play(paper), 1);
    ASSERT_EQ(scissor.play(scissor), 0);
    ASSERT_EQ(scissor.play(stone), -1);
}
// win = 1, draw = 0, loose = -1
TEST(CachipunClass, TestPaper) {
    Scissor scissor;
    Paper paper;
    Stone stone;
    ASSERT_EQ(paper.play(scissor), -1);
    ASSERT_EQ(paper.play(paper), 0);
    ASSERT_EQ(paper.play(stone), 1);
}
```

Googletest - Example



```
// win = 1, draw = 0, loose = -1
TEST(CachipunClass, TestStone) {
    Scissor scissor;
    Paper paper;
    Stone stone;
    ASSERT_EQ(stone.play(scissor), 1);
    ASSERT_EQ(stone.play(paper),-1);
    ASSERT_EQ(stone.play(stone), 0);
}

// win = 1, draw = 0, loose = -1
TEST(CachipunClass, TestScissor) {
    Scissor scissor;
    Paper paper;
    Stone stone;
    ASSERT_EQ(scissor.play(paper), 1);
    ASSERT_EQ(scissor.play(scissor), 0);
    ASSERT_EQ(scissor.play(stone), -1);
}
// win = 1, draw = 0, loose = -1
TEST(CachipunClass, TestPaper) {
    Scissor scissor;
    Paper paper;
    Stone stone;
    ASSERT_EQ(paper.play(scissor), -1);
    ASSERT_EQ(paper.play(paper), 0);
    ASSERT_EQ(paper.play(stone), 1);
}
```

Data used by the tests. Assertions exercises operations on the data

Fixture

It is very common to have *data used by unit tests*

As soon as tests are not trivial, you will need to have some data, ready to be used by the tests

In the World of testing, this data is called *fixture*

Using Googletest, a fixture is defined as a class, and define data that can be used in many different tests

```cpp
class CachipunTest : public ::testing::Test {
protected:
    Scissor scissor;
    Paper paper;
    Stone stone;
};

// win = 1, draw = 0, loose = -1
TEST_F(CachipunTest, TestStone) {
    ASSERT_EQ(stone.play(scissor), 1);
    ASSERT_EQ(stone.play(paper),-1);
    ASSERT_EQ(stone.play(stone), 0);
}

// win = 1, draw = 0, loose = -1
TEST_F(CachipunTest, TestScissor) {
    ASSERT_EQ(scissor.play(paper), 1);
    ASSERT_EQ(scissor.play(scissor), 0);
    ASSERT_EQ(scissor.play(stone), -1);
}

// win = 1, draw = 0, loose = -1
TEST_F(CachipunTest, TestPaper) {
    ASSERT_EQ(paper.play(scissor), -1);
    ASSERT_EQ(paper.play(paper), 0);
    ASSERT_EQ(paper.play(stone), 1);
}
```

```
class CachipunTest : public ::testing::Test {
protected:
    Scissor scissor;
    Paper paper;
    Stone stone;
};

// win = 1, draw = 0, loose = -1
TEST_F(CachipunTest, TestStone) {
    ASSERT_EQ(stone.play(scissor), 1);
    ASSERT_EQ(stone.play(paper),-1);
    ASSERT_EQ(stone.play(stone), 0);
}

// win = 1, draw = 0, loose = -1
TEST_F(CachipunTest, TestScissor) {
    ASSERT_EQ(scissor.play(paper), 1);
    ASSERT_EQ(scissor.play(scissor), 0);
    ASSERT_EQ(scissor.play(stone), -1);
}

// win = 1, draw = 0, loose = -1
TEST_F(CachipunTest, TestPaper) {
    ASSERT_EQ(paper.play(scissor), -1);
    ASSERT_EQ(paper.play(paper), 0);
    ASSERT_EQ(paper.play(stone), 1);
}
```

CachipunTest is a fixture, defined as a subclass of ::testing::Test

Initializing the fixture

The fixture for the cachipun example does not require any initialization

However, *initializing a fixture may involve a sequence of non-trivial steps*

Googletest offers the necessary to *initialize the fixture*

```
class FileSystemTest : public ::testing::Test {
protected:
    void SetUp() override {
        emptyFS = new FileSystem();
        fs =  new FileSystem();
        d1 = new Directory("directory1");
        d2 = new Directory("directory2");
        textFile = new TextFile("file.txt", "Hello World!");

        int content[4] = {65, 66, 67, 68};
        binaryFile = new BinaryFile("binary.bin", content, 4);

        d1->add(d2);
        d1->add(textFile);
        d1->add(binaryFile);

        fs->add(d1);
    }

    FileSystem *emptyFS, *fs;
    Directory *d1, *d2;
    TextFile *textFile;
    BinaryFile *binaryFile;
};
```

```
class FileSystemTest : public ::testing::Test {
protected:
    void SetUp() override {
        emptyFS = new FileSyst
        fs =  new FileSystem()
        d1 = new Directory("di
        d2 = new Directory("di
        textFile = new TextFile("file.txt", "Hello World!");

        int content[4] = {65, 66, 67, 68};
        binaryFile = new BinaryFile("binary.bin", content, 4);

        d1->add(d2);
        d1->add(textFile);
        d1->add(binaryFile);

        fs->add(d1);
    }

    FileSystem *emptyFS, *fs;
    Directory *d1, *d2;
    TextFile *textFile;
    BinaryFile *binaryFile;
};
```

Ensure you are really doing an override

```
TEST_F(FileSystemTest, getSize) {
    ASSERT_EQ(0, emptyFS->getSize());
    ASSERT_EQ(16, fs->getSize());
}

TEST_F(FileSystemTest, getNumberOfFiles) {
    ASSERT_EQ(0, emptyFS->getNumberOfFiles());
    ASSERT_EQ(2, fs->getNumberOfFiles());

    Directory d("another directory");
    d.add(new TextFile("another file", "bonjour"));
    fs->add(&d);
    ASSERT_EQ(3, fs->getNumberOfFiles());
}

TEST_F(FileSystemTest, getNumberOfDirectories) {
    ASSERT_EQ(1, emptyFS->getNumberOfDirectories());
    ASSERT_EQ(3, fs->getNumberOfDirectories());

    Directory d("another directory");
    d.add(new TextFile("another file", "bonjour"));
    fs->add(&d);
    ASSERT_EQ(4, fs->getNumberOfDirectories());
}
```

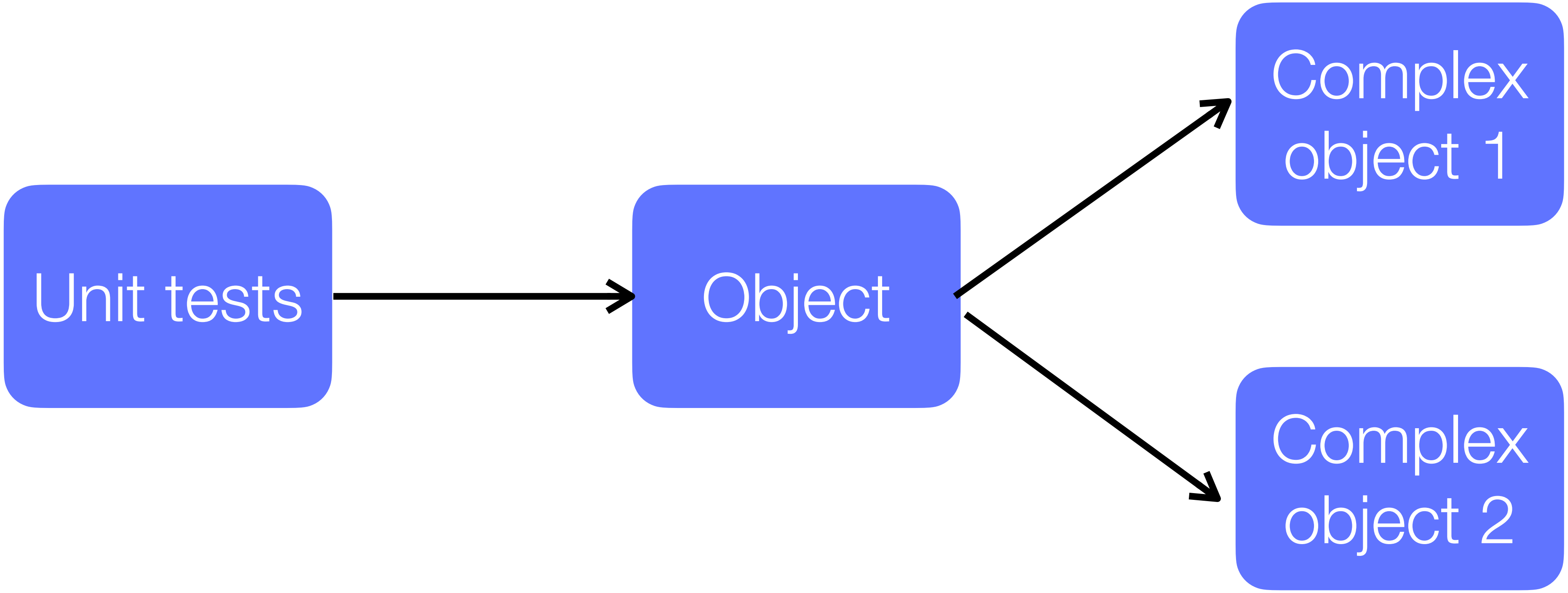
Explicit Fixture

Non-trivial tests must have a fixture, and it happens that the same fixture can be used in many different tests

Having a class Fixture help *reducing code duplication*, and r*educe the complexity of the tests*

Having simple and clear tests is important because unit tests are often considered as a *“living” documentation*

Testing scenario

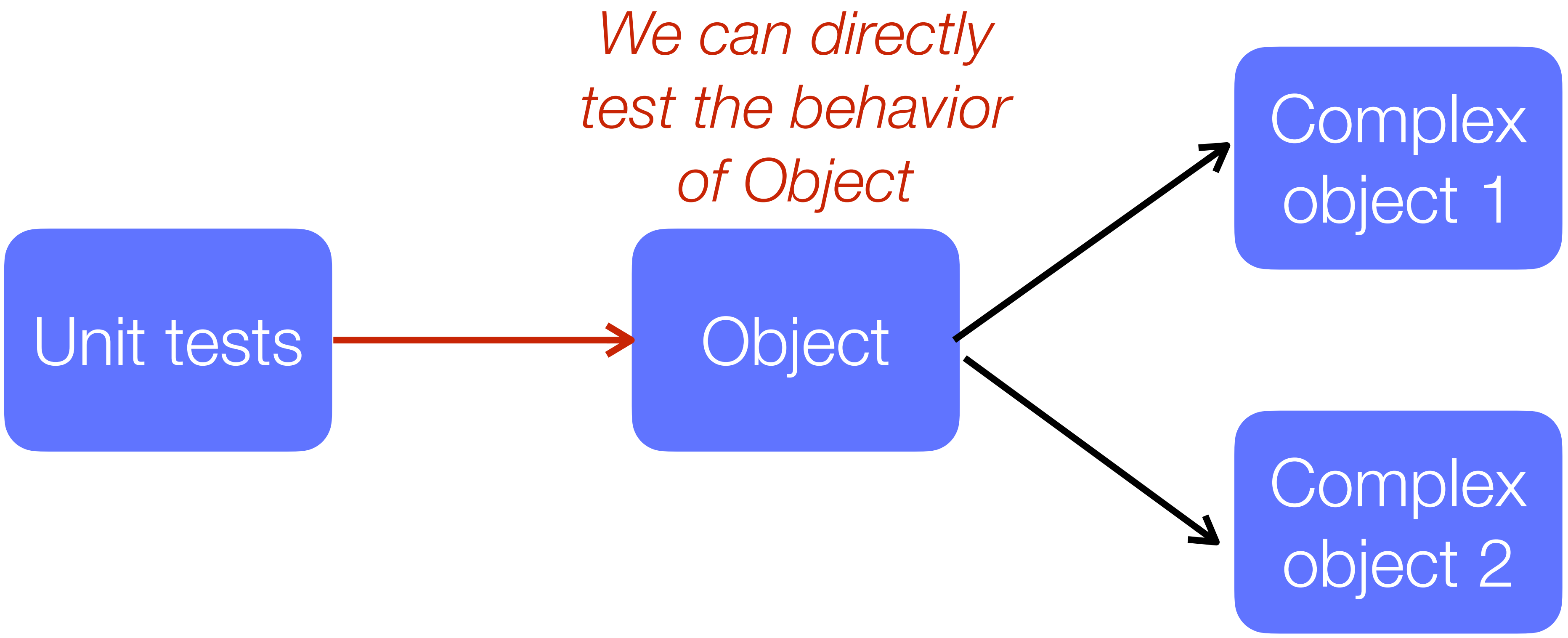

Unit tests

Object

Complex object 1

Complex object 2

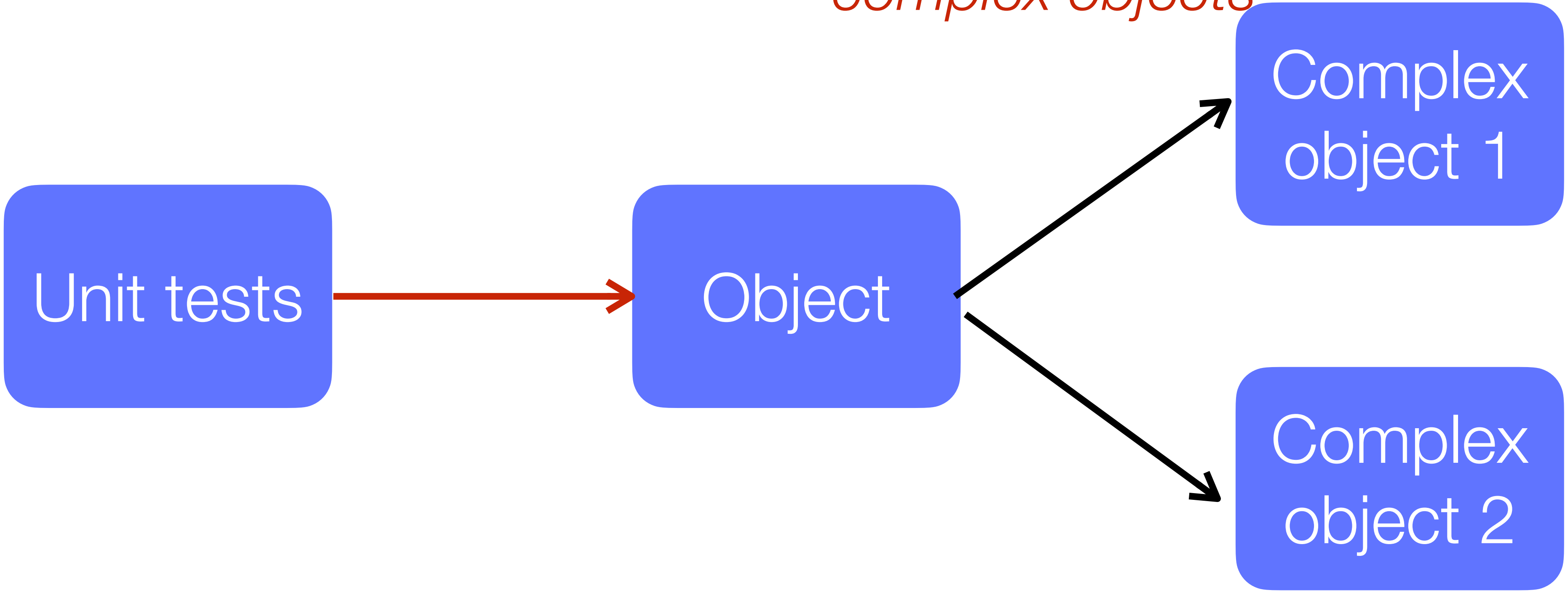
Testing scenario



We can directly test the behavior of Object

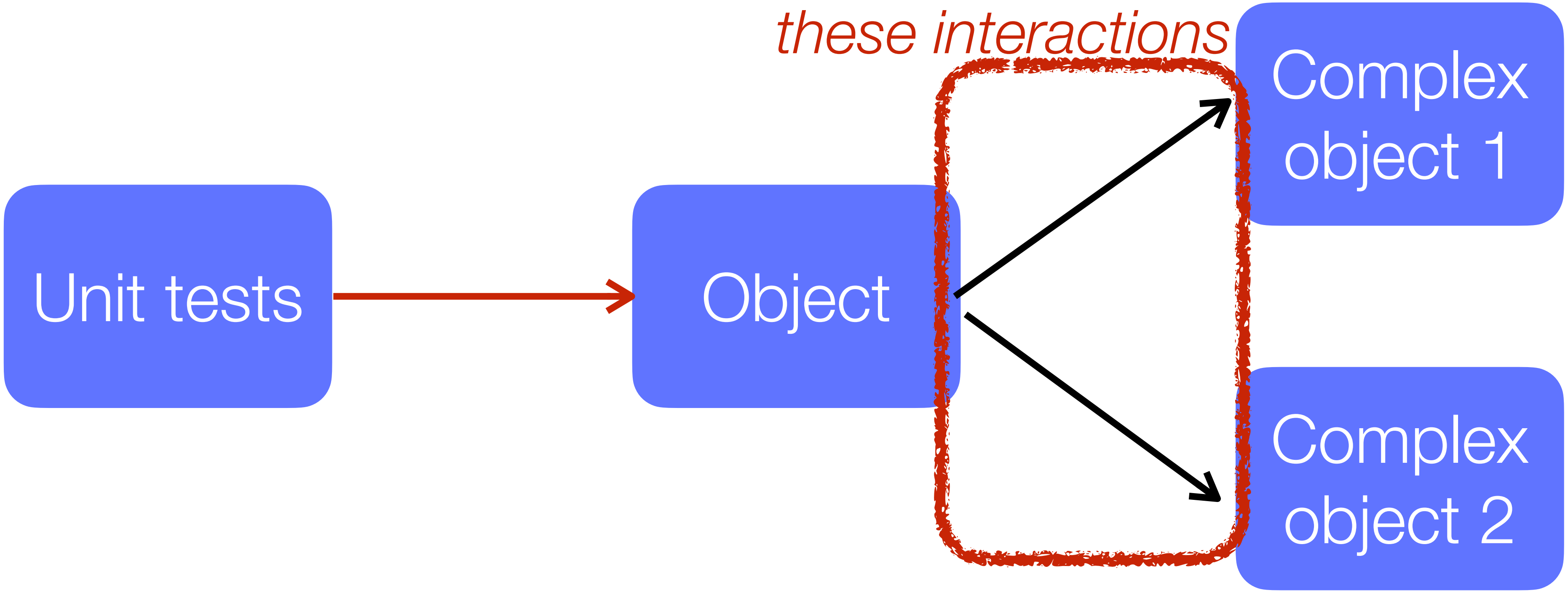
Testing scenario

However, we cannot test the interaction with the complex objects



Testing scenario

Mock testing is about testing these interactions



Mocking

Mocking is a testing technique used to *isolate complex object behavior*

Mock objects *simulate the behavior of real objects*

A test will now test whether the mocked objects *are used properly*

Part of the test verifies that the *mock was used correctly*

Assertions are about how the code under test is *interacting with other system modules*

gMock is part of Googletest

```
class MockDirectory : public Directory {
public:
    MockDirectory(string aName) : Directory(aName) {}
    MOCK_METHOD(vector<Item*>, getItems, (), (override));
    MOCK_METHOD(void, add, (Item* anItem), (override));
    MOCK_METHOD(void, accept, (Visitor* v), (override));
    MOCK_METHOD(int, getSize, (), (override));
};

TEST_F(FileSystemTest, testingDirectory) {
    MockDirectory d("another directory");
    EXPECT_CALL(d, getSize()).Times(AtLeast(1));
    d.add(new TextFile("another file", "bonjour"));
    fs->add(&d);
    fs->getSize();

    EXPECT_CALL(d, accept).Times(Exactly(1));
    fs->getNumberOfFiles();
}
```

```
class MockDirectory : public Directory {
public:
    MockDirectory(string aName) : Directory(aName) {}
    MOCK_METHOD(vector<Item*>, getItems, (), (override));
    MOCK_METHOD(void, add, (Item* anItem), (override));
    MOCK_METHOD(void, accept, (Visitor* v), (override));
    MOCK_METHOD(int, getSize, (), (override));
};
```

```
TEST_F(FileSystemTest, testingDirectory) {
    MockDirectory d("another directory");
```

Define the `MockDirectory` class. The mock class needs to defines mock methods for each virtual function of `Directory`

The mock directory is hooked into the filesystem we created in the fixture.

```
};

TEST_F(FileSystemTest, testingDirectory) {
    MockDirectory d("another directory");
    EXPECT_CALL(d, getSize()).Times(AtLeast(1));
    d.add(new TextFile("another file", "bonjour"));
    fs->add(&d);
    fs->getSize();

    EXPECT_CALL(d, accept).Times(Exactly(1));
    fs->getNumberOfFiles();
}
```

Rules may be defined to describe part of behavior of mocked objects. E.g., getSize() is called at least once, and accept is called exactly 1 time.

```
};

TEST_F(FileSystemTest, testingDirectory) {
    MockDirectory d("another directory");
    EXPECT_CALL(d, getSize()).Times(AtLeast(1));
    d.add(new TextFile("another file", "bonjour"));
    fs->add(&d);
    fs->getSize();

    EXPECT_CALL(d, accept).Times(Exactly(1));
    fs->getNumberOfFiles();
}
```

Rules may be defined to describe part of behavior of mocked objects. E.g., `getSize()` is called at least once, and accept is called exactly 1 time.

```
};

TEST_F(FileSystemTest, testingDirectory) {
    MockDirectory d("another directory");
    EXPECT_CALL(d, getSize()).Times(AtLeast(1));
    d.add(new TextFile("another file", "bonjour"));
    fs->add(&d);
    fs->getSize();

    EXPECT_CALL(d, accept).Times(Exactly(1));
    fs->getNumberOfFiles();
}
```

The call of `getNumberOfFiles()` creates a visitor and make it run

Mocking

The previous example shows a case in which we use gMock to test:

- The method getSize() is called exactly once on a Directory when calling getSize() on the filesystem
- The method accept(…) is called exactly once when calling getNumberOfFiles() on the file system.

These two tests are difficult to express without a mocking framework

Mocking - Making CMakeLists happy

Do not forget to add `gmock_main` in the `test/CMakeLists.txt`

in the `target_link_libraries` section

Mocking a class

Longer description of gMock may be found on:

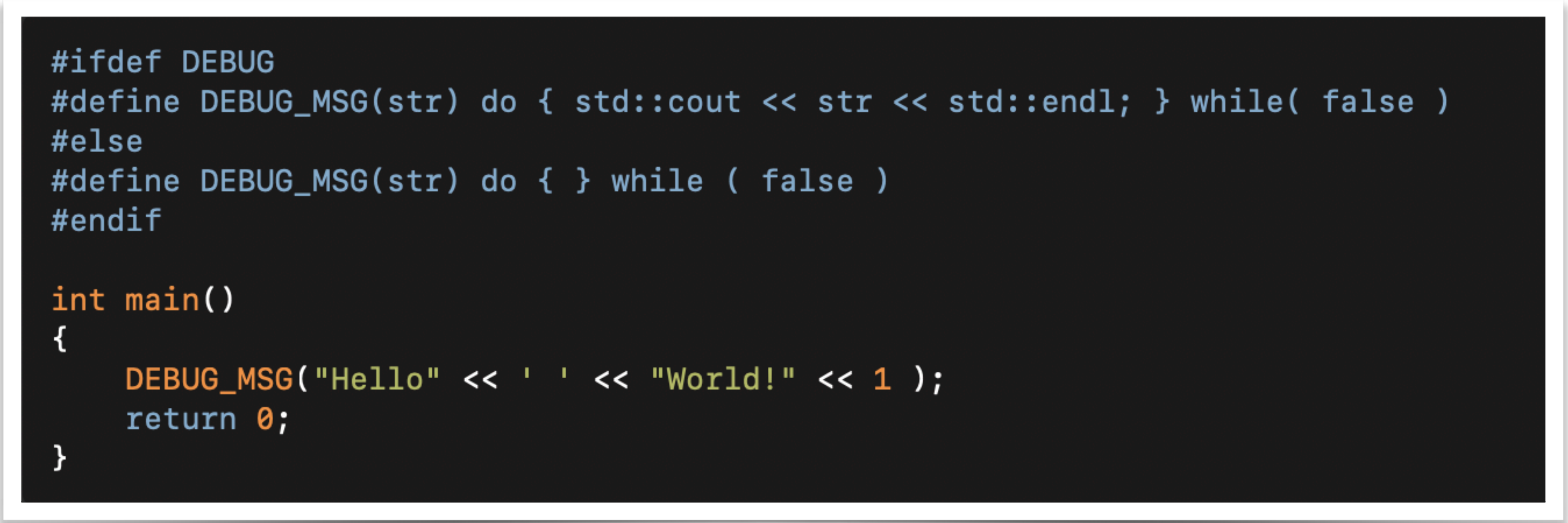
https://google.github.io/googletest/gmock_for_dummies.html

gMock can be used with Boost without any problem

- just make sure that mock objects are *not* copied

Roadmap

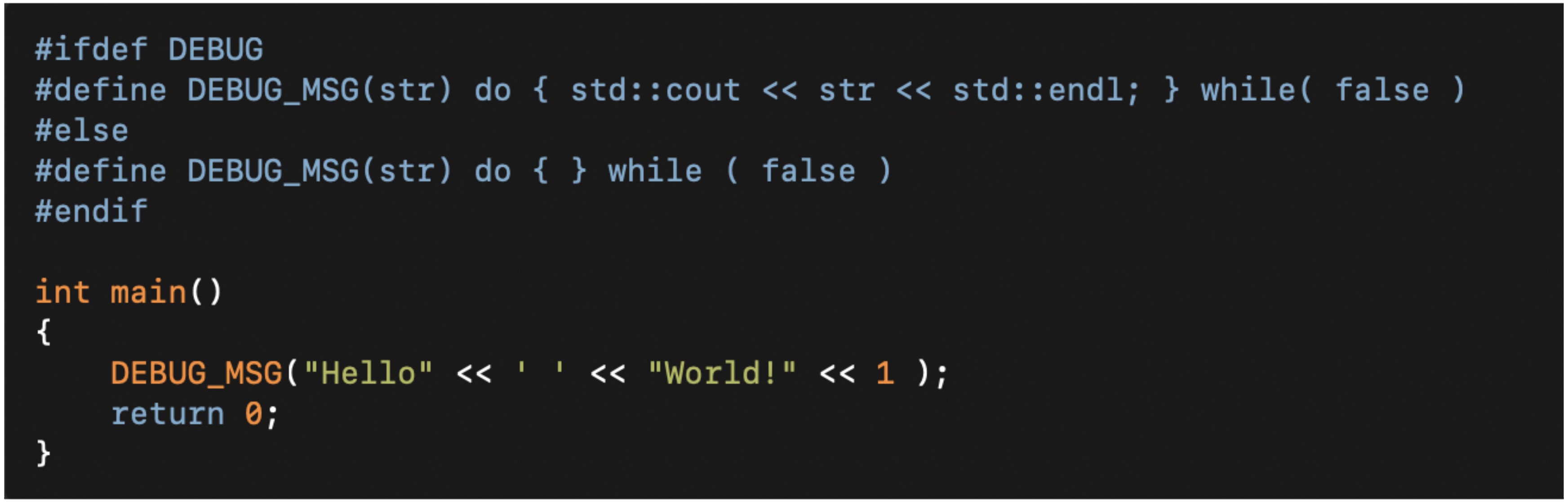
1. Tests (fixture, mock)
2. **Debugging**



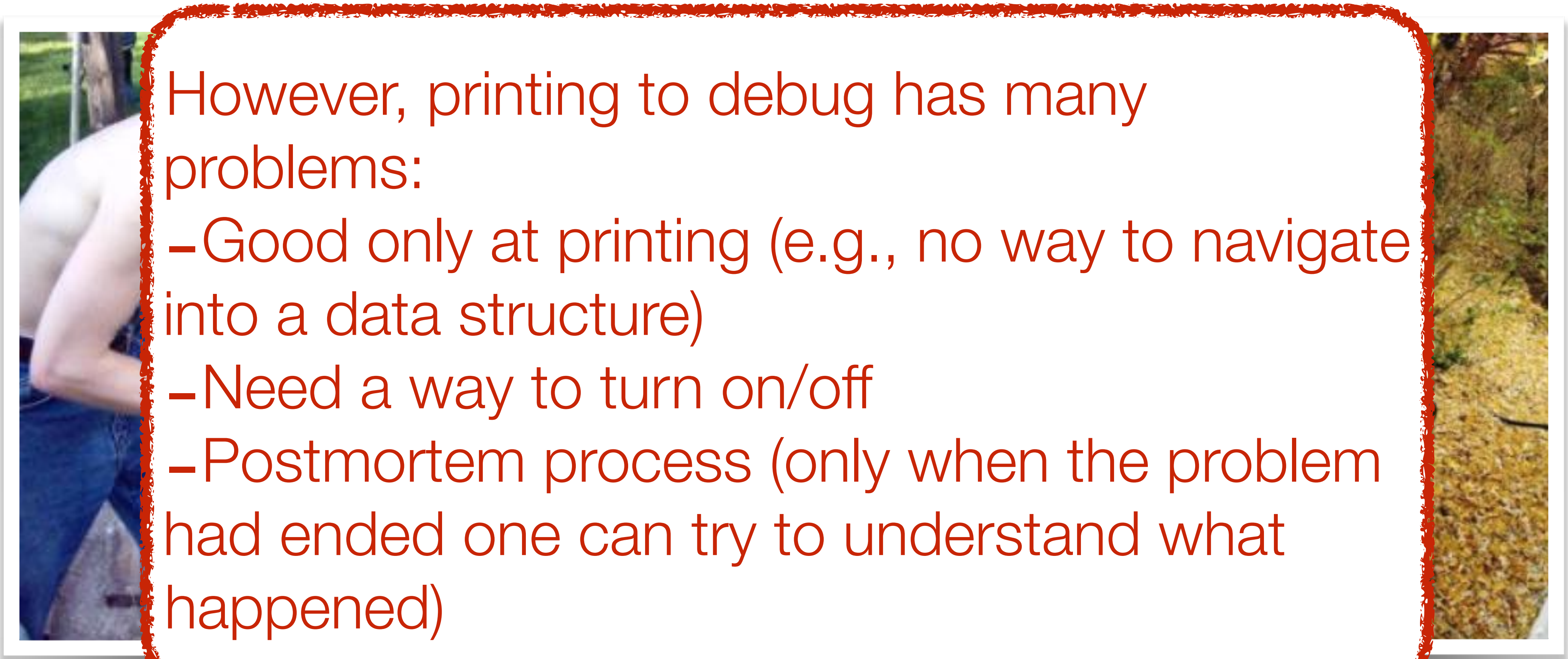
```
#ifdef DEBUG
#define DEBUG_MSG(str) do { std::cout << str << std::endl; } while( false )
#else
#define DEBUG_MSG(str) do { } while ( false )
#endif

int main()
{
    DEBUG_MSG("Hello" << ' ' << "World!" << 1 );
    return 0;
}
```

Good thing about printing:

- Very easy to use
- No need to learn a new tool
- Very flexible (one can print anything)

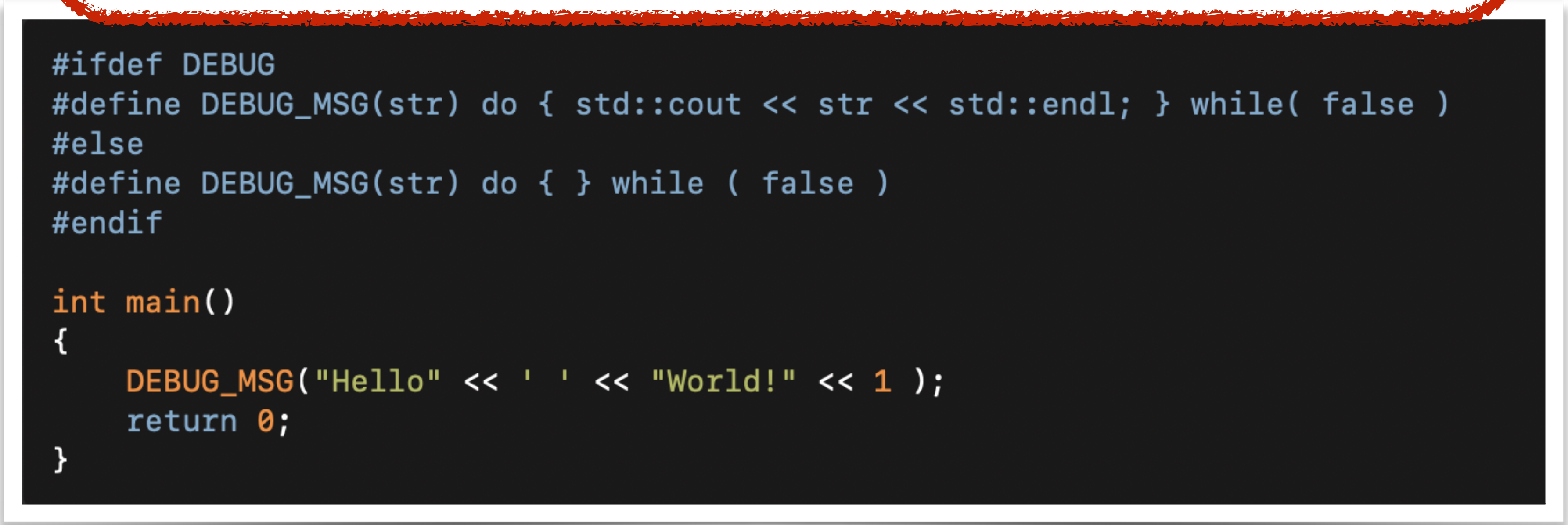


```
#ifdef DEBUG
#define DEBUG_MSG(str) do { std::cout << str << std::endl; } while( false )
#else
#define DEBUG_MSG(str) do { } while ( false )
#endif

int main()
{
    DEBUG_MSG("Hello" << ' ' << "World!" << 1 );
    return 0;
}
```



However, printing to debug has many problems:

- Good only at printing (e.g., no way to navigate into a data structure)
- Need a way to turn on/off
- Postmortem process (only when the problem had ended one can try to understand what happened)



```
#ifdef DEBUG
#define DEBUG_MSG(str) do { std::cout << str << std::endl; } while( false )
#else
#define DEBUG_MSG(str) do { } while ( false )
#endif

int main()
{
    DEBUG_MSG("Hello" << ' ' << "World!" << 1 );
    return 0;
}
```

Debugging

The standard debuggers for C++ are called `gdb` and `lldb`

Usable from the command line

But a UI will make you significantly faster to use

Most programming environments uses `gdb` or `lldb` underneath

Debugging

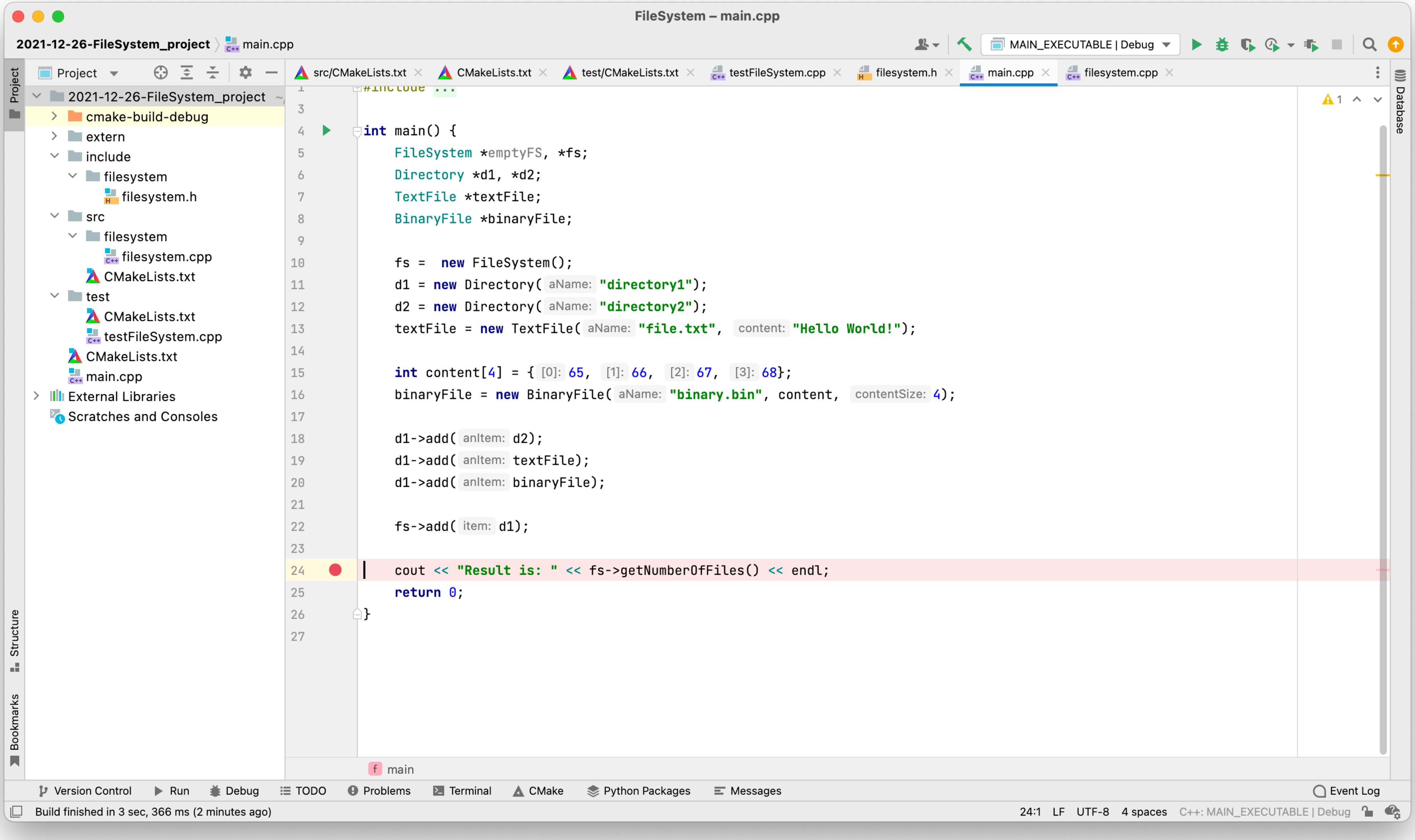
A debugger offer:

Breakpoints to tell the program under run to suspend

Inspector of the heap and the runtime callstack

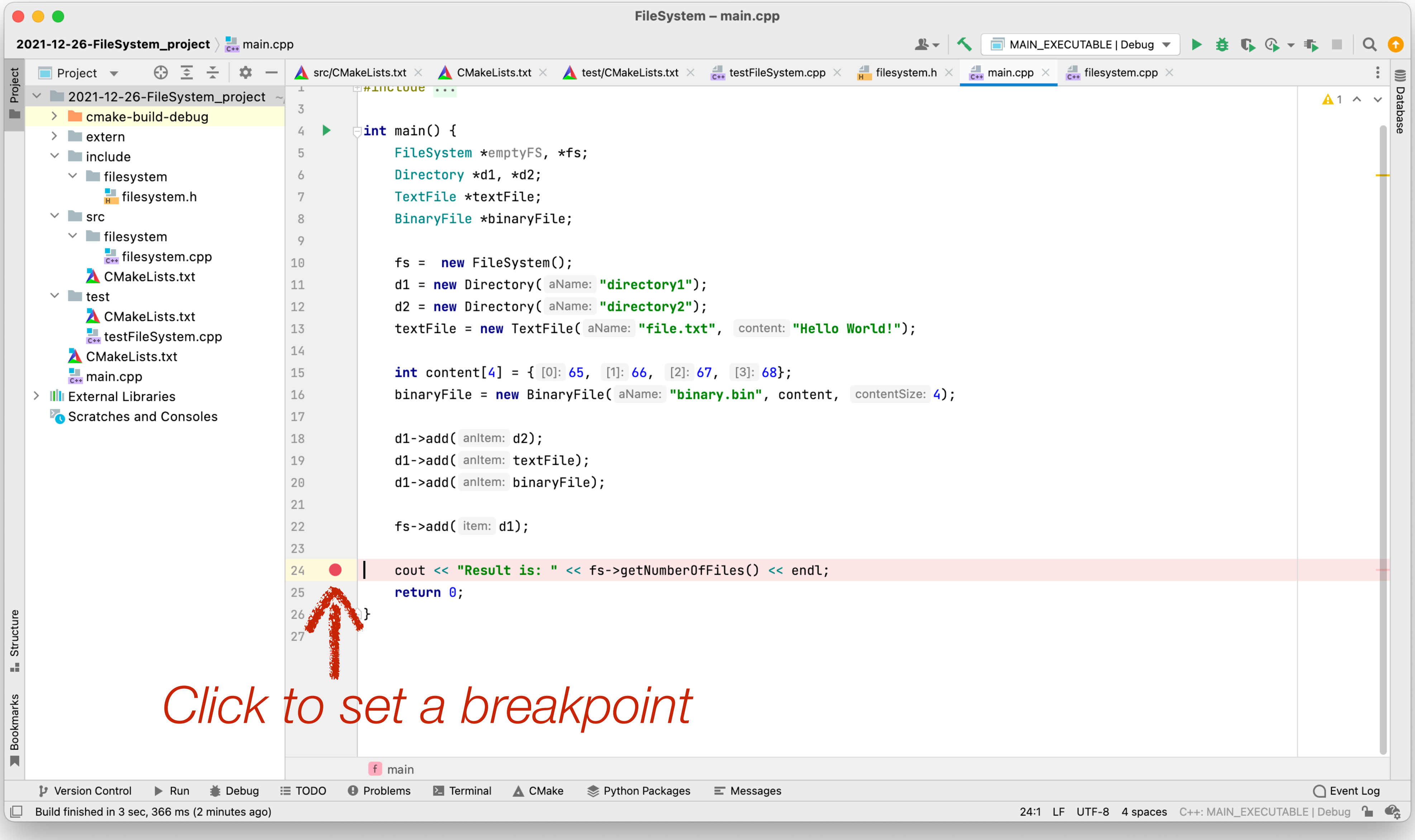
Operations to manually execute statements

Watcher to see the value of different instructions



```
#include ...

int main() {
    FileSystem *emptyFS, *fs;
    Directory *d1, *d2;
    TextFile *textFile;
    BinaryFile *binaryFile;

    fs =  new FileSystem();
    d1 = new Directory( aName: "directory1");
    d2 = new Directory( aName: "directory2");
    textFile = new TextFile( aName: "file.txt",  content: "Hello World!");

    int content[4] = { [0]: 65,  [1]: 66,  [2]: 67,  [3]: 68};
    binaryFile = new BinaryFile( aName: "binary.bin", content,  contentSize: 4);

    d1->add( anItem: d2);
    d1->add( anItem: textFile);
    d1->add( anItem: binaryFile);

    fs->add( item: d1);

    cout << "Result is: " << fs->getNumberOfFiles() << endl;
    return 0;
}
```

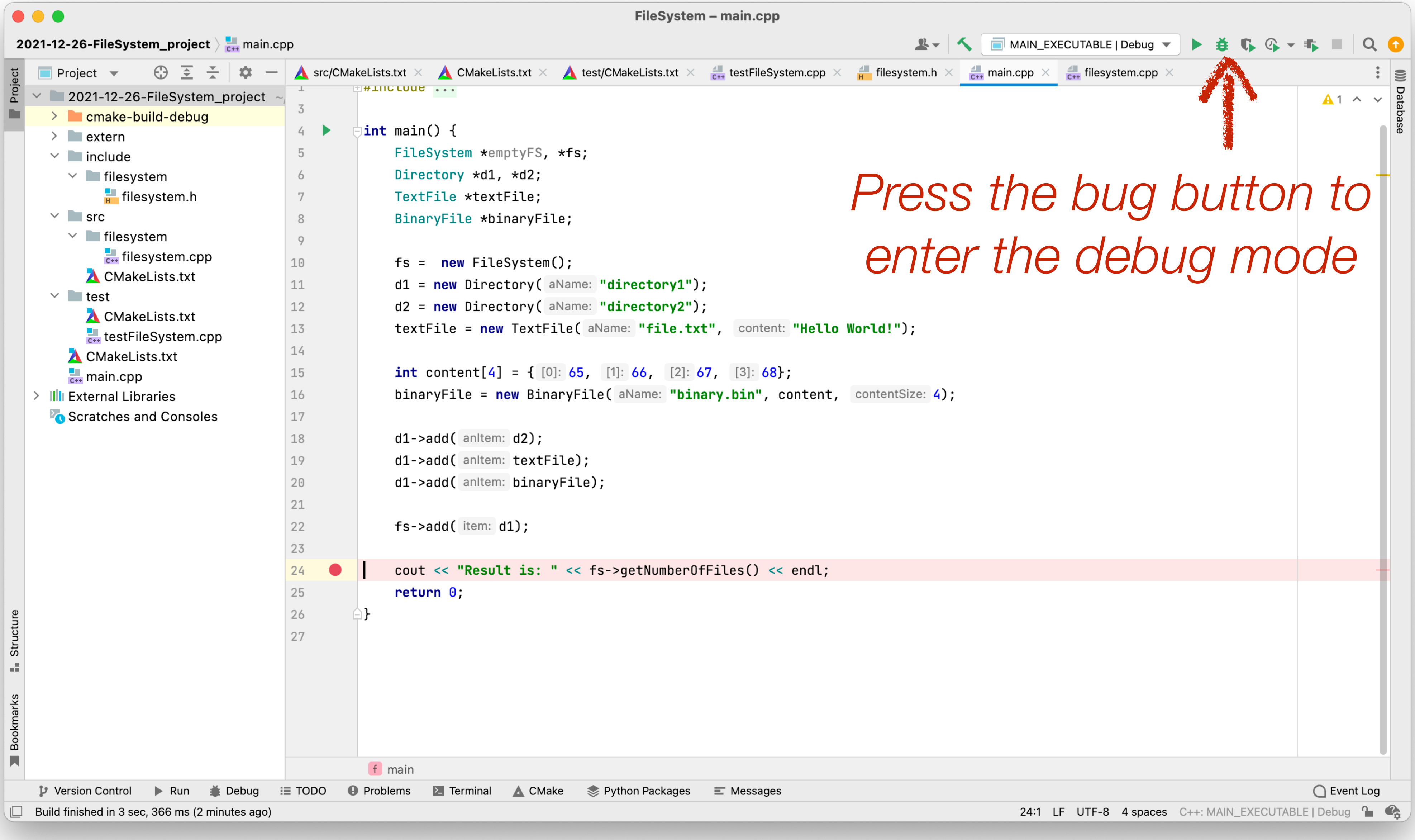
Build finished in 3 sec, 366 ms (2 minutes ago)



```
#include ...

int main() {
    FileSystem *emptyFS, *fs;
    Directory *d1, *d2;
    TextFile *textFile;
    BinaryFile *binaryFile;

    fs =  new FileSystem();
    d1 = new Directory( aName: "directory1");
    d2 = new Directory( aName: "directory2");
    textFile = new TextFile( aName: "file.txt",  content: "Hello World!");

    int content[4] = { [0]: 65,  [1]: 66,  [2]: 67,  [3]: 68};
    binaryFile = new BinaryFile( aName: "binary.bin", content,  contentSize: 4);

    d1->add( anItem: d2);
    d1->add( anItem: textFile);
    d1->add( anItem: binaryFile);

    fs->add( item: d1);

    cout << "Result is: " << fs->getNumberOfFiles() << endl;
    return 0;
}
```

Build finished in 3 sec, 366 ms (2 minutes ago)

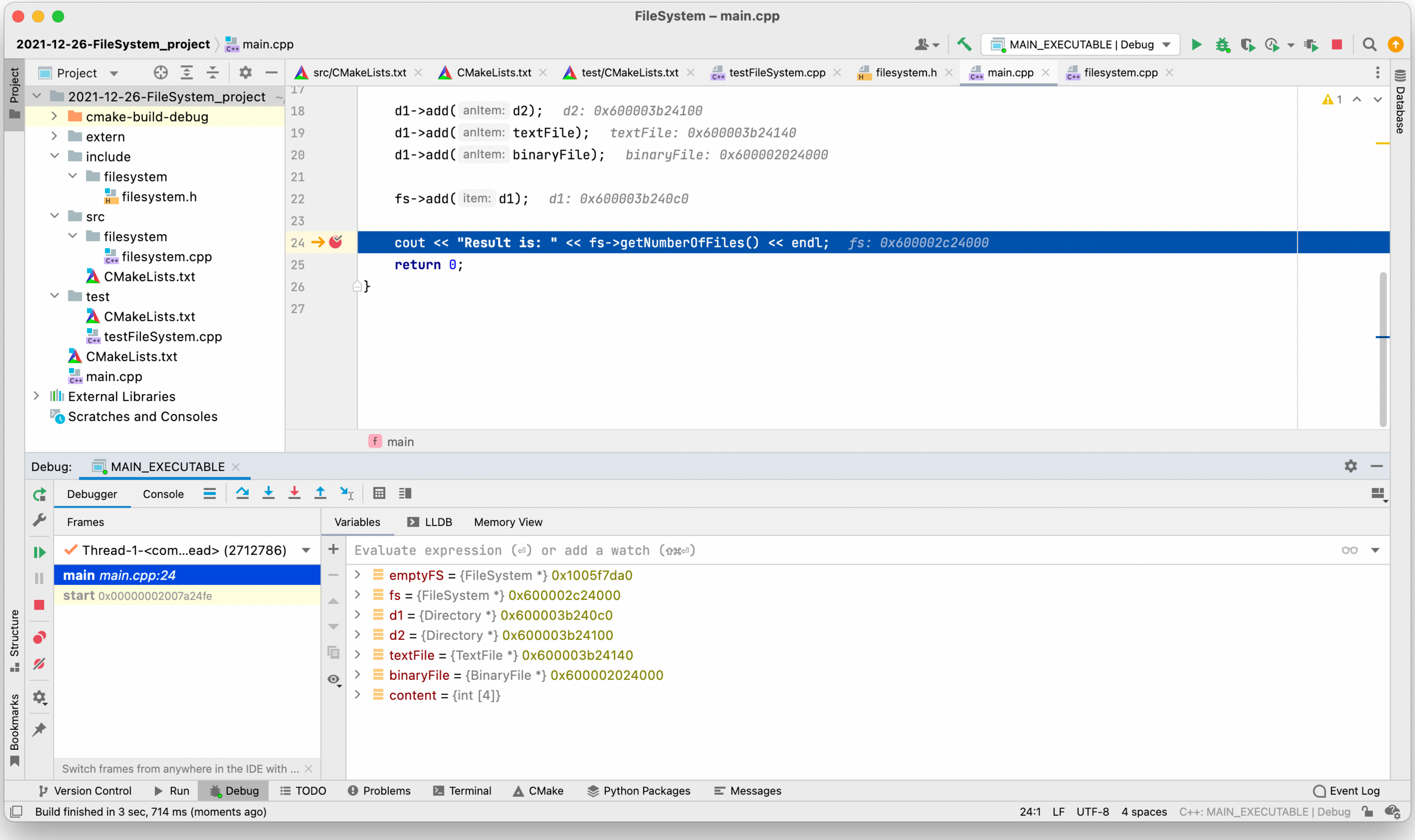
Click to set a breakpoint



```cpp
#include ...

int main() {
    FileSystem *emptyFS, *fs;
    Directory *d1, *d2;
    TextFile *textFile;
    BinaryFile *binaryFile;

    fs =  new FileSystem();
    d1 = new Directory( aName: "directory1");
    d2 = new Directory( aName: "directory2");
    textFile = new TextFile( aName: "file.txt",  content: "Hello World!");

    int content[4] = { [0]: 65,  [1]: 66,  [2]: 67,  [3]: 68};
    binaryFile = new BinaryFile( aName: "binary.bin", content,  contentSize: 4);

    d1->add( anItem: d2);
    d1->add( anItem: textFile);
    d1->add( anItem: binaryFile);

    fs->add( item: d1);

    cout << "Result is: " << fs->getNumberOfFiles() << endl;
    return 0;
}
```

Build finished in 3 sec, 366 ms (2 minutes ago)

Press the bug button to enter the debug mode



```
    d1->add( anItem: d2);   d2: 0x600003b24100
    d1->add( anItem: textFile);   textFile: 0x600003b24140
    d1->add( anItem: binaryFile);   binaryFile: 0x600002024000

    fs->add( item: d1);   d1: 0x600003b240c0

    cout << "Result is: " << fs->getNumberOfFiles() << endl;   fs: 0x600002c24000
    return 0;
}
```

Thread-1-<com...ead> (2712786)

main main.cpp:24

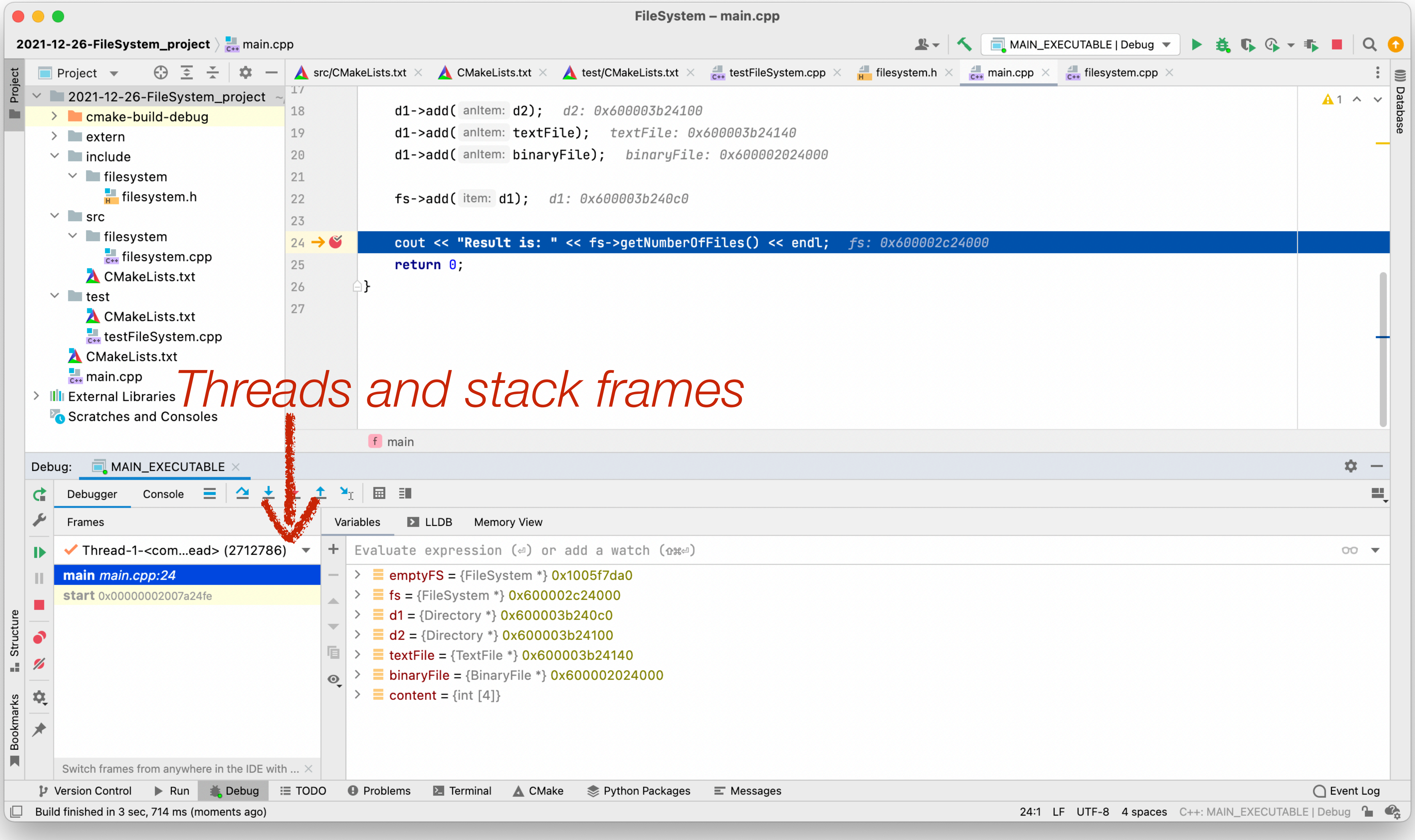
start 0x00000002007a24fe

Switch frames from anywhere in the IDE with ...

Evaluate expression (⏎) or add a watch (⇧⌘⏎)

```
> emptyFS = {FileSystem *} 0x1005f7da0
> fs = {FileSystem *} 0x600002c24000
> d1 = {Directory *} 0x600003b240c0
> d2 = {Directory *} 0x600003b24100
> textFile = {TextFile *} 0x600003b24140
> binaryFile = {BinaryFile *} 0x600002024000
> content = {int [4]}
```

Build finished in 3 sec, 714 ms (moments ago)   24:1 LF UTF-8 4 spaces C++: MAIN_EXECUTABLE | Debug



```
    d1->add( anItem: d2);   d2: 0x600003b24100
    d1->add( anItem: textFile);   textFile: 0x600003b24140
    d1->add( anItem: binaryFile);   binaryFile: 0x600002024000

    fs->add( item: d1);   d1: 0x600003b240c0

    cout << "Result is: " << fs->getNumberOfFiles() << endl;   fs: 0x600002c24000
    return 0;
}
```

main

Threads and stack frames

Thread-1-<com...ead> (2712786)

main main.cpp:24

start 0x00000002007a24fe

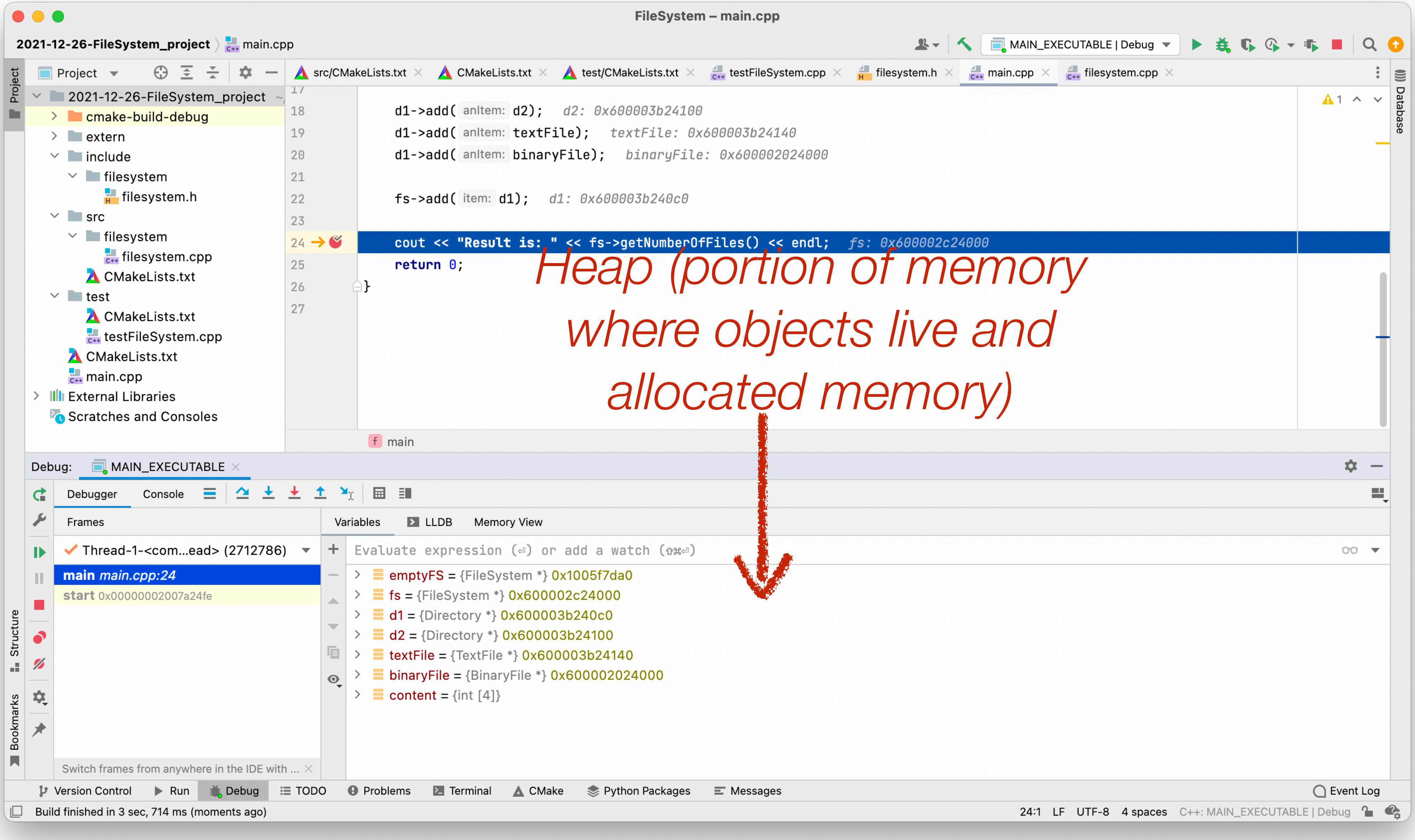
Switch frames from anywhere in the IDE with ...

Evaluate expression (⏎) or add a watch (⇧⌘⏎)

```
> emptyFS = {FileSystem *} 0x1005f7da0
> fs = {FileSystem *} 0x600002c24000
> d1 = {Directory *} 0x600003b240c0
> d2 = {Directory *} 0x600003b24100
> textFile = {TextFile *} 0x600003b24140
> binaryFile = {BinaryFile *} 0x600002024000
> content = {int [4]}
```

Build finished in 3 sec, 714 ms (moments ago)

24:1 LF UTF-8 4 spaces C++: MAIN_EXECUTABLE | Debug

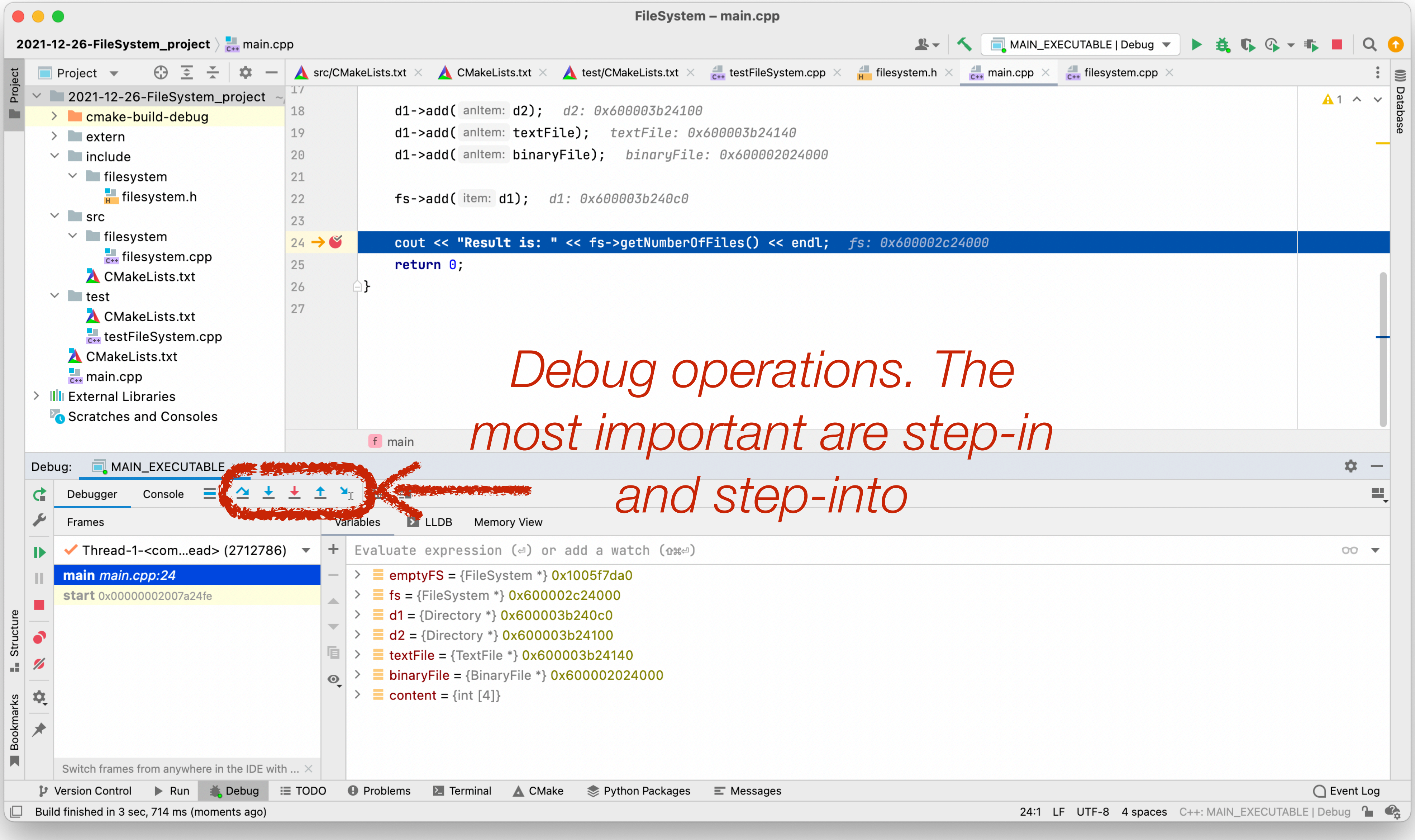


```
    d1->add( anItem: d2);   d2: 0x600003b24100
    d1->add( anItem: textFile);   textFile: 0x600003b24140
    d1->add( anItem: binaryFile);   binaryFile: 0x600002024000

    fs->add( item: d1);   d1: 0x600003b240c0

    cout << "Result is: " << fs->getNumberOfFiles() << endl;   fs: 0x600002c24000
    return 0;
}
```

Heap (portion of memory where objects live and allocated memory)

```
emptyFS = {FileSystem *} 0x1005f7da0
fs = {FileSystem *} 0x600002c24000
d1 = {Directory *} 0x600003b240c0
d2 = {Directory *} 0x600003b24100
textFile = {TextFile *} 0x600003b24140
binaryFile = {BinaryFile *} 0x600002024000
content = {int [4]}
```

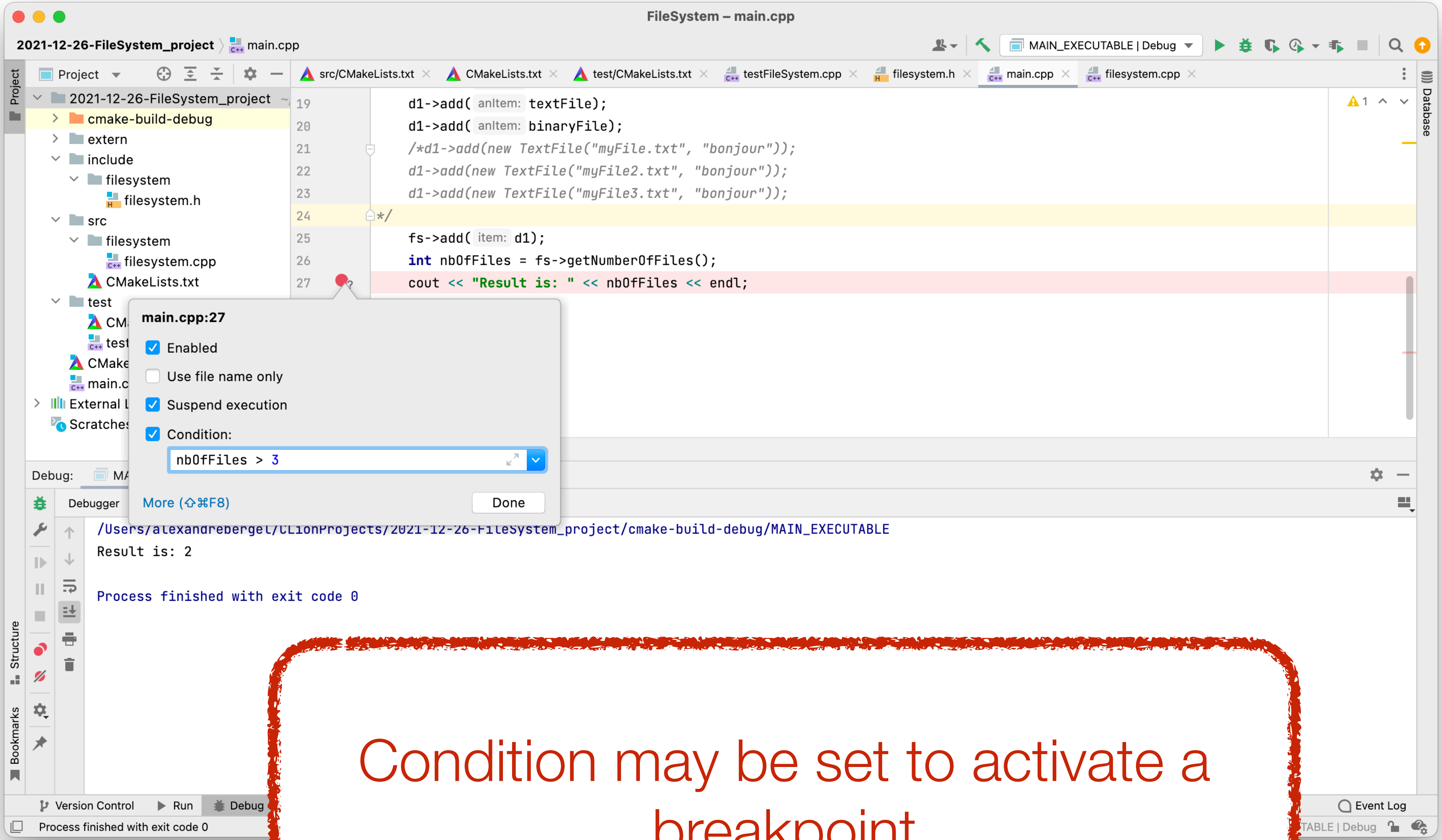
Build finished in 3 sec, 714 ms (moments ago)



```
    d1->add( anItem: d2);   d2: 0x600003b24100
    d1->add( anItem: textFile);   textFile: 0x600003b24140
    d1->add( anItem: binaryFile);   binaryFile: 0x600002024000

    fs->add( item: d1);   d1: 0x600003b240c0

    cout << "Result is: " << fs->getNumberOfFiles() << endl;   fs: 0x600002c24000
    return 0;
}
```

```
emptyFS = {FileSystem *} 0x1005f7da0
fs = {FileSystem *} 0x600002c24000
d1 = {Directory *} 0x600003b240c0
d2 = {Directory *} 0x600003b24100
textFile = {TextFile *} 0x600003b24140
binaryFile = {BinaryFile *} 0x600002024000
content = {int [4]}
```

Debug operations. The most important are step-in and step-into



```
    d1->add( anItem: textFile);
    d1->add( anItem: binaryFile);
    /*d1->add(new TextFile("myFile.txt", "bonjour"));
    d1->add(new TextFile("myFile2.txt", "bonjour"));
    d1->add(new TextFile("myFile3.txt", "bonjour"));
*/
    fs->add( item: d1);
    int nbOfFiles = fs->getNumberOfFiles();
    cout << "Result is: " << nbOfFiles << endl;
```

main.cpp:27

☑ Enabled

□ Use file name only

☑ Suspend execution

☑ Condition:

```
nbOfFiles > 3
```

More (⇧⌘F8) Done

```
/Users/alexandrebergel/CLionProjects/2021-12-26-FileSystem_project/cmake-build-debug/MAIN_EXECUTABLE
Result is: 2

Process finished with exit code 0
```

Process finished with exit code 0

Condition may be set to activate a breakpoint

Breakpoint

Different kind of breakpoints are supported in CLion:

- Unconditional breakpoint
- Breakpoint with condition
- Breakpoint when exception are raised

Support for full customization of the breakpoint (simply right click on a breakpoint)

Exercise (optional)

Add fixture and mocks in your tests

What you should know!

Explicitly defining a fixture is essential as soon as test are non-trivial

Mocking is an expressive way to check for some program invariant

The debugger must be your new friend, forever!

Can you answer these questions?

How to test for sequence in method calls using gMock?

License

Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)
You are free to:
- Share: copy and redistribute the material in any medium or format
- Adapt: remix, transform, and build upon the material for any purpose, even commercially

The licensor cannot revoke these freedoms as long as you follow the license terms

Attribution: you must give appropriate credit

ShareAlike: if you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original

Complete license: https://creativecommons.org/licenses/by-sa/4.0/

Original version of this lecture from Oscar Nierstrasz, Uni - Bern dcc

CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

www.dcc.uchile.cl

/ DCCUCHILE