# CUnit

# Outline of This Lab

1. Factorial Example

2. Install Cunit

3. Execute testing Projects Successfully

# CUnit

C-based open source framework for writing and running  unit tests.

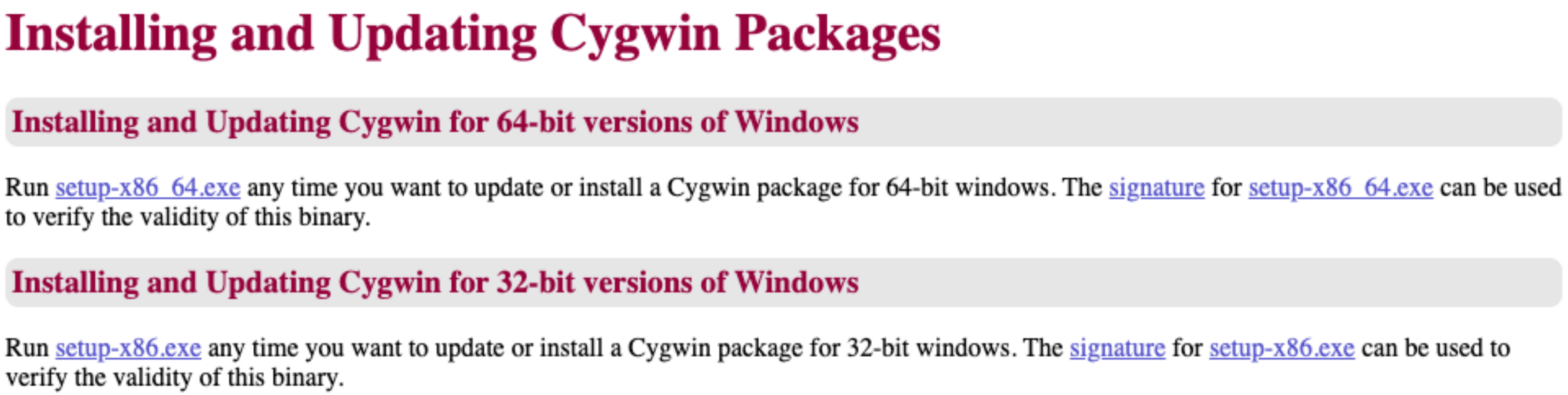
**Provides:**

- Test suites to group test cases
- Assertions for testing expected results within a test case
- Automates the execution of test cases showing the results

# CUnit Installation On Windows

# Install Cygwin

Install Cygwin using the link provided at `https://cygwin.com/install.html`



## Installing and Updating Cygwin Packages

### Installing and Updating Cygwin for 64-bit versions of Windows

Run setup-x86_64.exe any time you want to update or install a Cygwin package for 64-bit windows. The signature for setup-x86_64.exe can be used to verify the validity of this binary.

### Installing and Updating Cygwin for 32-bit versions of Windows

Run setup-x86.exe any time you want to update or install a Cygwin package for 32-bit windows. The signature for setup-x86.exe can be used to verify the validity of this binary.

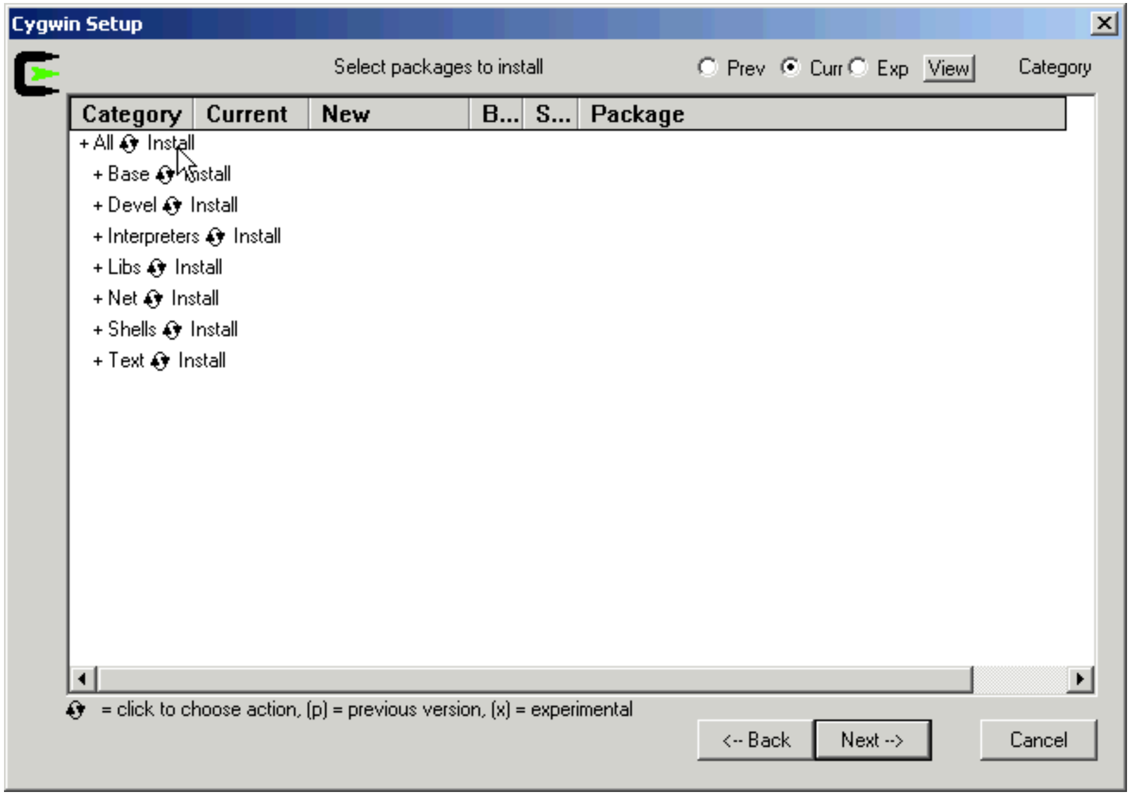
For 64bit Win select **setup-x86_64.exe**

For 32bit Win select **setup-x86.exe**

https://cygwin.com/install.html

# Choosing Packages

- You should be able to see packages as classified into categories.
- Select and Install CUnit in the category Libs



Cygwin Setup

Select packages to install

○ Prev ● Curr ○ Exp View Category

| Category | Current | New | B... | S... | Package |
|---|---|---|---|---|---|
| + All Install | | | | | |
| + Base Install | | | | | |
| + Devel Install | | | | | |
| + Interpreters Install | | | | | |
| + Libs Install | | | | | |
| + Net Install | | | | | |
| + Shells Install | | | | | |
| + Text Install | | | | | |

= click to choose action, (p) = previous version, (x) = experimental

# CUnit Installation On MacOS

# Install CUnit on MacOS

If you do not have brew installed, open a Terminal and execute the following command:

```
ruby -e "$(curl -fsSL https://
raw.githubusercontent.com/Homebrew/install/
master/install)" < /dev/null 2> /dev/null
```

Then execute the following command:

```
brew install cunit
```

# CUnit Installation On Linux

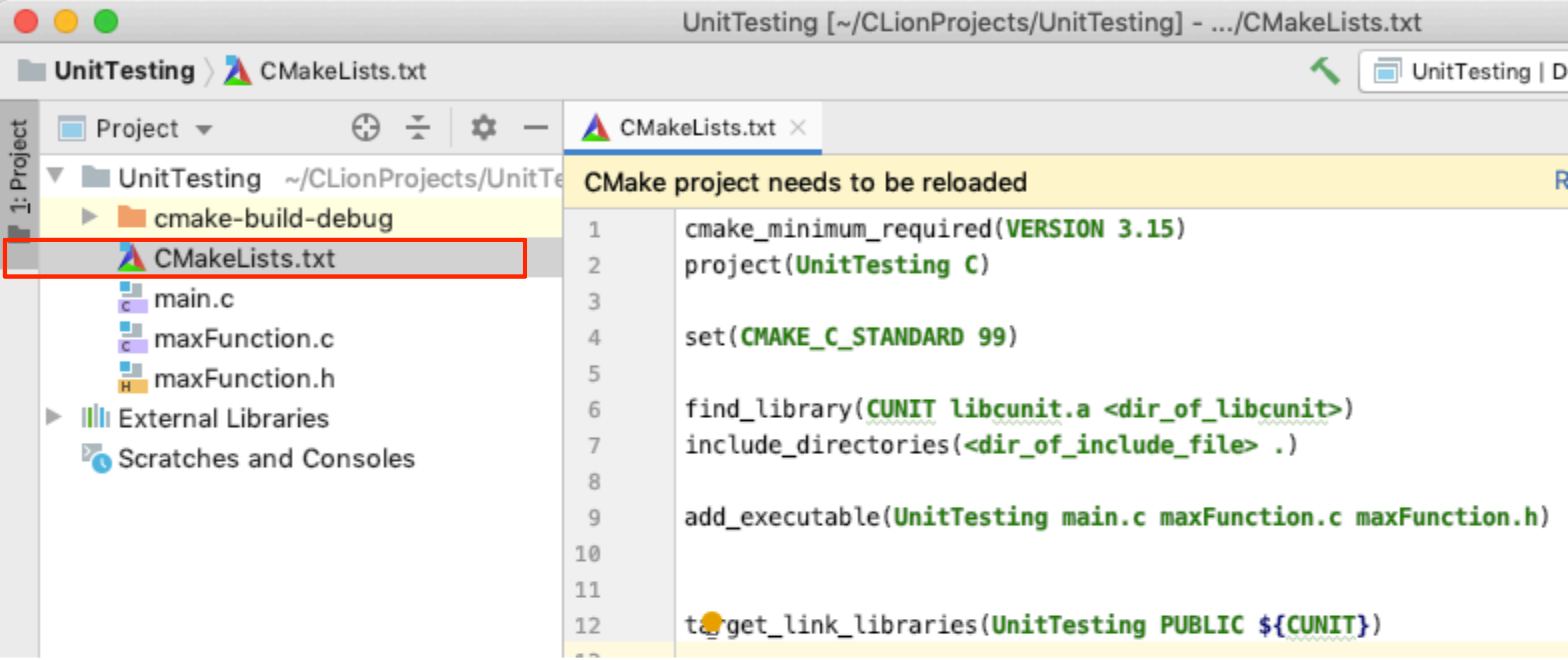
# Install CUnit on Linux

Open your terminal and execute the following commands:

```
sudo apt-get update
sudo apt-get install libcunit1
libcunit1-doc libcunit1-dev
```

# Configure CUnit

# Open One of the Testing Projects Provided On BrightSpace in Clion (Max Function Testing Example)

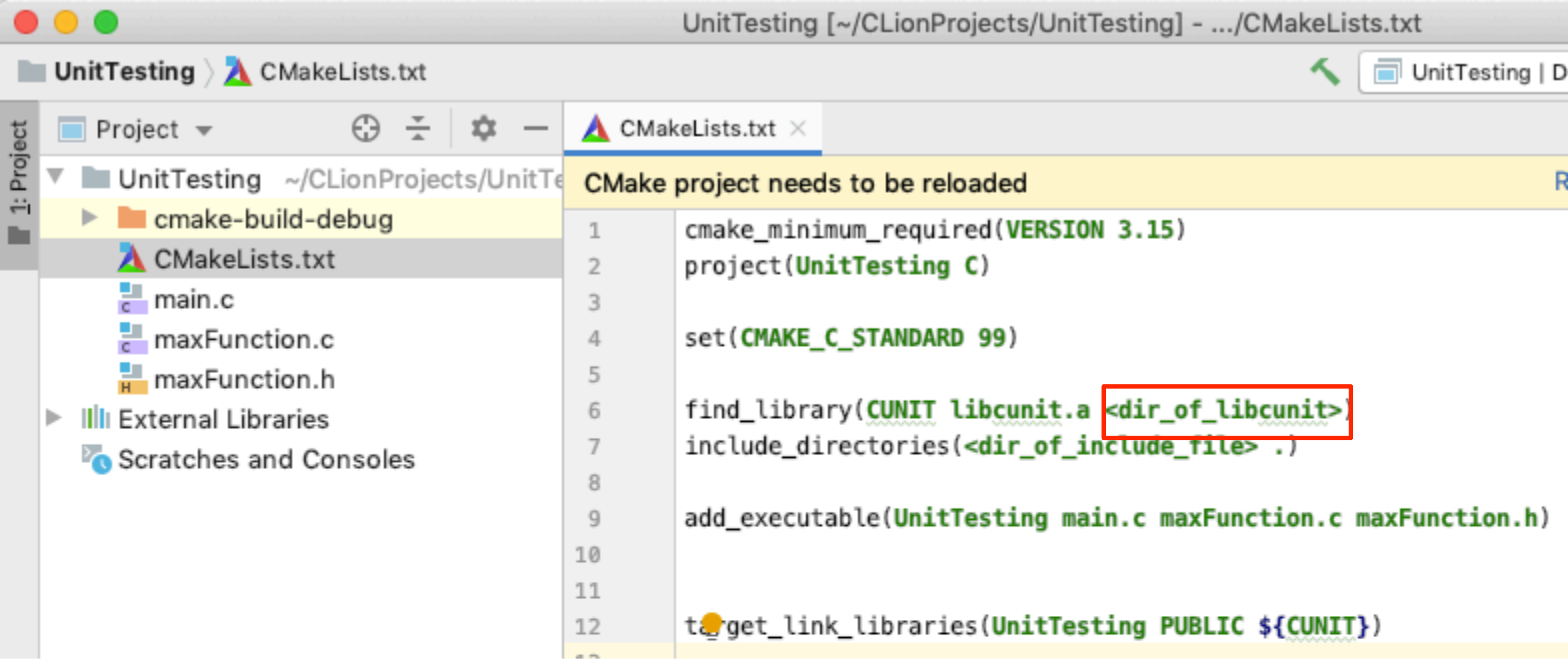


CMake project needs to be reloaded

```
cmake_minimum_required(VERSION 3.15)
project(UnitTesting C)

set(CMAKE_C_STANDARD 99)

find_library(CUNIT libcunit.a <dir_of_libcunit>)
include_directories(<dir_of_include_file> .)

add_executable(UnitTesting main.c maxFunction.c maxFunction.h)


target_link_libraries(UnitTesting PUBLIC ${CUNIT})
```

Open file *CMakeLists.txt*

# Set-up the directory where your CUnit installation is present



CMake project needs to be reloaded

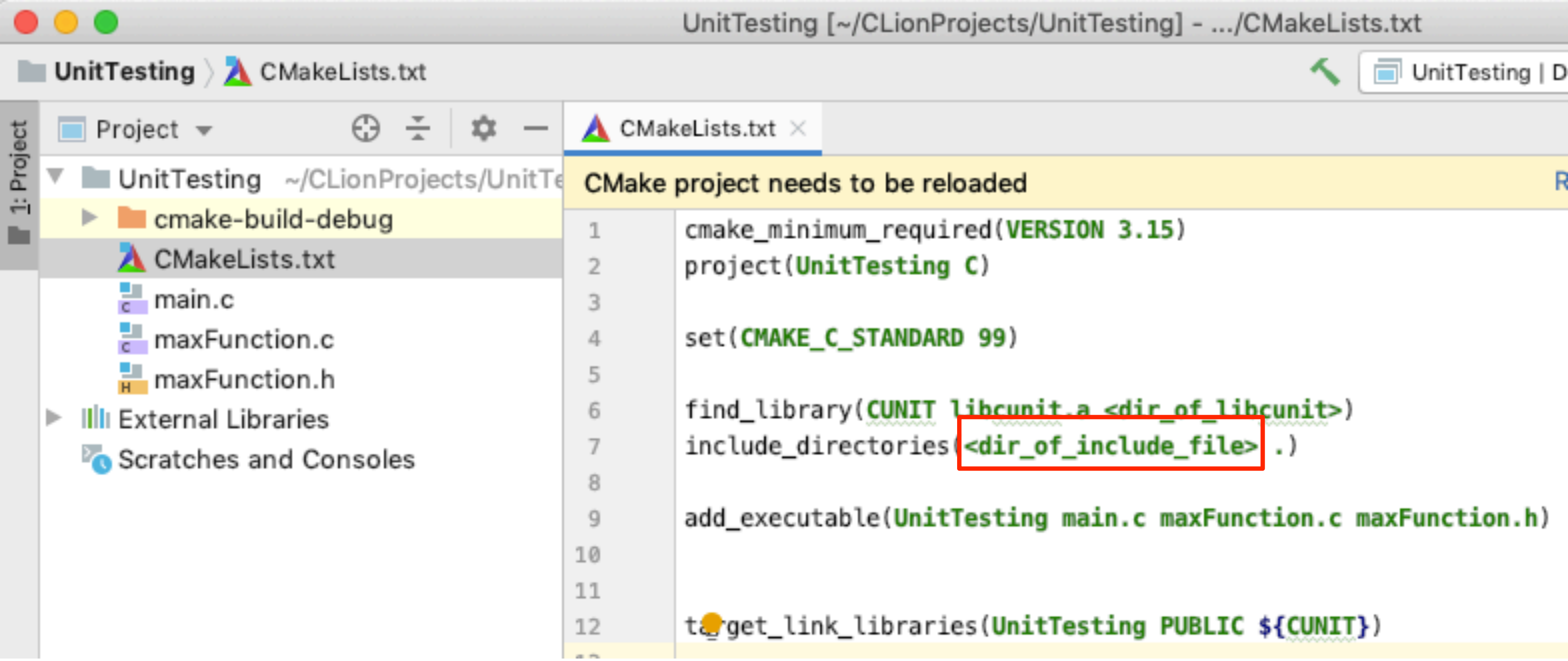
```
cmake_minimum_required(VERSION 3.15)
project(UnitTesting C)

set(CMAKE_C_STANDARD 99)

find_library(CUNIT libcunit.a <dir_of_libcunit>)
include_directories(<dir_of_include_file> .)

add_executable(UnitTesting main.c maxFunction.c maxFunction.h)


target_link_libraries(UnitTesting PUBLIC ${CUNIT})
```

Replace the string *<dir_of_libcunit>* with the directory where *libcunit.a* is located. For example:

On a Mac: /usr/local/Cellar/cunit/2.1-3/lib

On Linux: /usr/lib/x86_64-linux-gnu/

# Set-up the directory where the CUnit header files are located



CMake project needs to be reloaded

```
cmake_minimum_required(VERSION 3.15)
project(UnitTesting C)

set(CMAKE_C_STANDARD 99)

find_library(CUNIT libcunit.a <dir_of_libcunit>)
include_directories(<dir_of_include_file> .)

add_executable(UnitTesting main.c maxFunction.c maxFunction.h)


target_link_libraries(UnitTesting PUBLIC ${CUNIT})
```

Replace the string *<dir_of_include_file>* with the directory of the CUnit header files. For example:

On MacOS: /usr/local/Cellar/cunit/2.1-3/include/CUnit

On Linux: /usr/include/CUnit/

# Now Try to Run The Examples

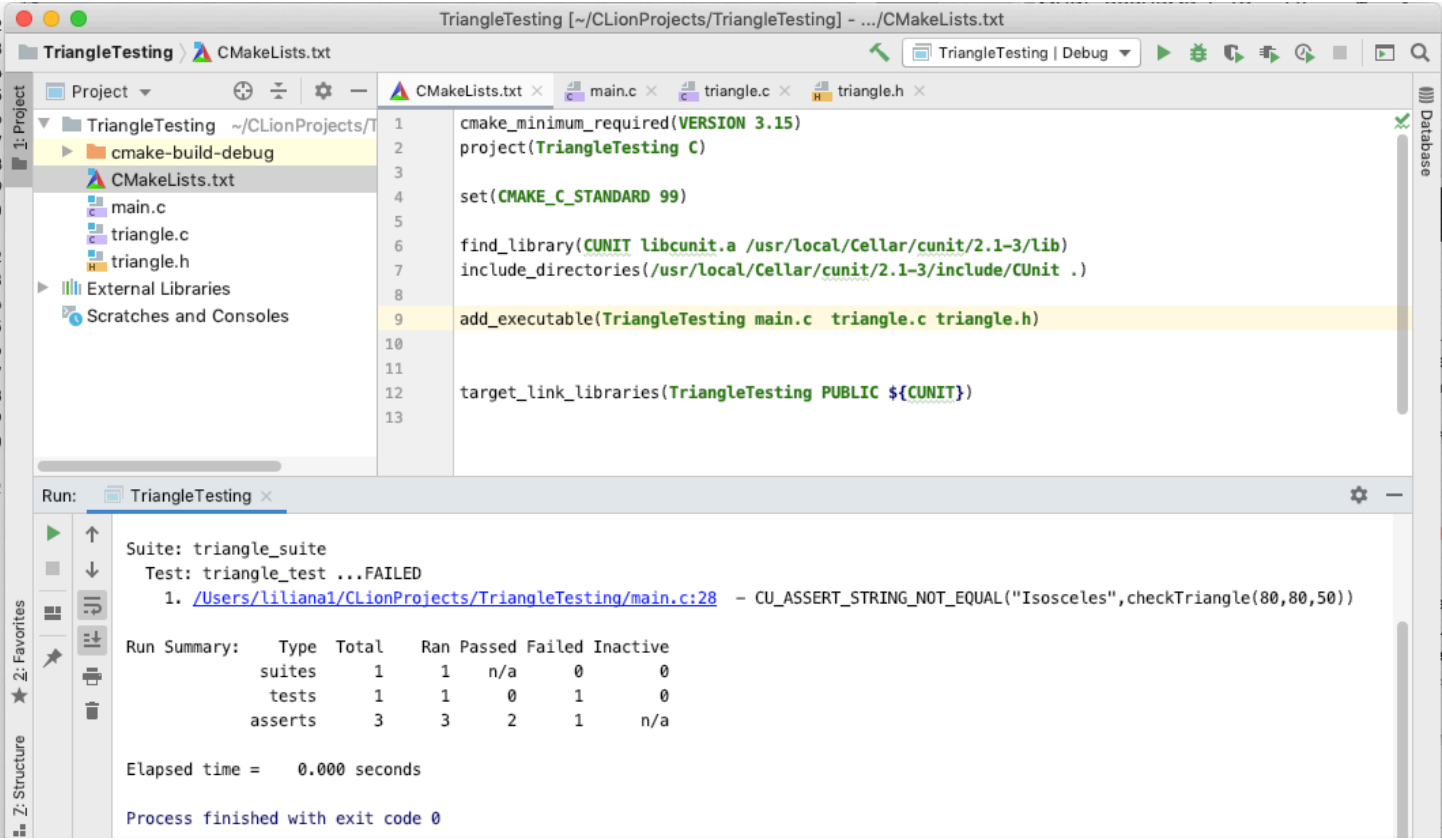
## On Brightspace there are 2 Projects:

- MaxFunction Testing Example
- Triangle Testing Example

## What you need to do

1. Import the projects on CLion and run them successfully (remember to set up CUnit dependencies)
2. Try the factorial example (described in the rest of the slides)

# Successful Execution of TriangleTesting Project



```
cmake_minimum_required(VERSION 3.15)
project(TriangleTesting C)

set(CMAKE_C_STANDARD 99)

find_library(CUNIT libcunit.a /usr/local/Cellar/cunit/2.1-3/lib)
include_directories(/usr/local/Cellar/cunit/2.1-3/include/CUnit .)

add_executable(TriangleTesting main.c  triangle.c triangle.h)


target_link_libraries(TriangleTesting PUBLIC ${CUNIT})
```

```
Suite: triangle_suite
  Test: triangle_test ...FAILED
    1. /Users/liliana1/CLionProjects/TriangleTesting/main.c:28  - CU_ASSERT_STRING_NOT_EQUAL("Isosceles",checkTriangle(80,80,50))

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests      1      1      0      1        0
             asserts      3      3      2      1      n/a

Elapsed time =    0.000 seconds

Process finished with exit code 0
```

# Exercise 1: Testing Factorial

# Method to Test

Suppose you want to verify a method

```
int factorial(int n)
```

This method calculates the factorial of a natural number **n**

# Method to Test

Suppose you want to verify a method

```
int factorial(int n)
```

This method calculates the factorial of a natural number **n**

- What inputs do you need to verify this method?

- What assertions you might need to verify?

# Method to Test

Suppose you want to verify a method

```
int factorial(int n)
```

This method calculates the factorial of a natural number **n**

- What inputs do you need to verify this method?
  - Normal values: 4
  - Boundary values: 0, 1

- What assertions you might need to verify?

# Method to Test

Suppose you want to verify a method

```
int factorial(int n)
```

This method calculates the factorial of a natural number **n**

- What inputs do you need to verify this method?
  - Normal values: 4
  - Boundary values: 0, 1

- What assertions you might need to verify?
  - Factorial of 4 is 24
  - Factorial of 0 and 1 is 1

# Create the Source Files Containing the Implementation of the Method to Test

- Open your IDE and create a C Project (e.g., Factorial)
- Inside this project define a c Source file factorial.c as follows

```
#include “factorial.h”
int factorial(int n) {
    //precondition: n >0
    int fact; //factorial of n
    int i; //to iterated between 1 and n
    /*calculates factorial of n */
    fact = 1;
    for(i=1; i<n; i++) {
        fact *= i;
    }
    return fact;
}
```

# Contain the Header File Including the Function Declaration

- Inside the Factorial project define a header file `factorial.h` containing the following line

```
int factorial(int n);
```

# Test the Factorial Function

- Now let’s test the Factorial function
  - Download the file testFactorial.c on Brightspace
  - Import testFactorial.c in the Factorial project

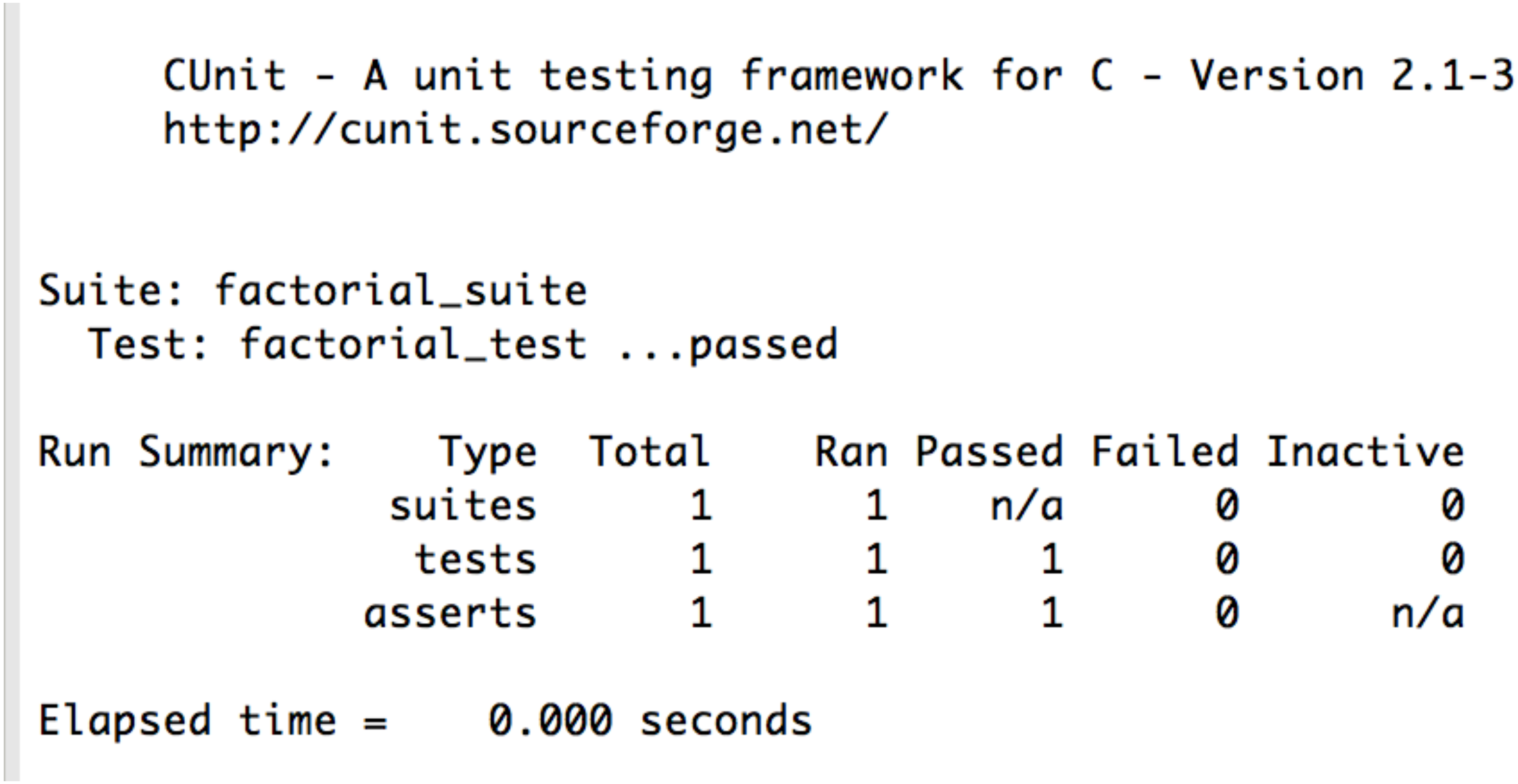
- Remember to configure the dependencies with CUnit libraries in your project

# Design the Test Cases for the Factorial Function

```
void factorial_testcase1(void){
    CU_ASSERT_EQUAL(factorial(0),1 );
    /* insert here 2 assertions necessary
     * to verify whether the factorial of 1 is 1
     * and whether the factorial of 4 is 24
     */
}
```

- The test case in testFactorial.c only has 1 assertion that verifies if the factorial of 0 is 1
- Run the CLion project

# The Test Case is Successful



```
     CUnit - A unit testing framework for C - Version 2.1-3
     http://cunit.sourceforge.net/


Suite: factorial_suite
  Test: factorial_test ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests      1      1      1      0        0
             asserts      1      1      1      0      n/a

Elapsed time =    0.000 seconds
```

# Design the Test Cases for the Factorial Function

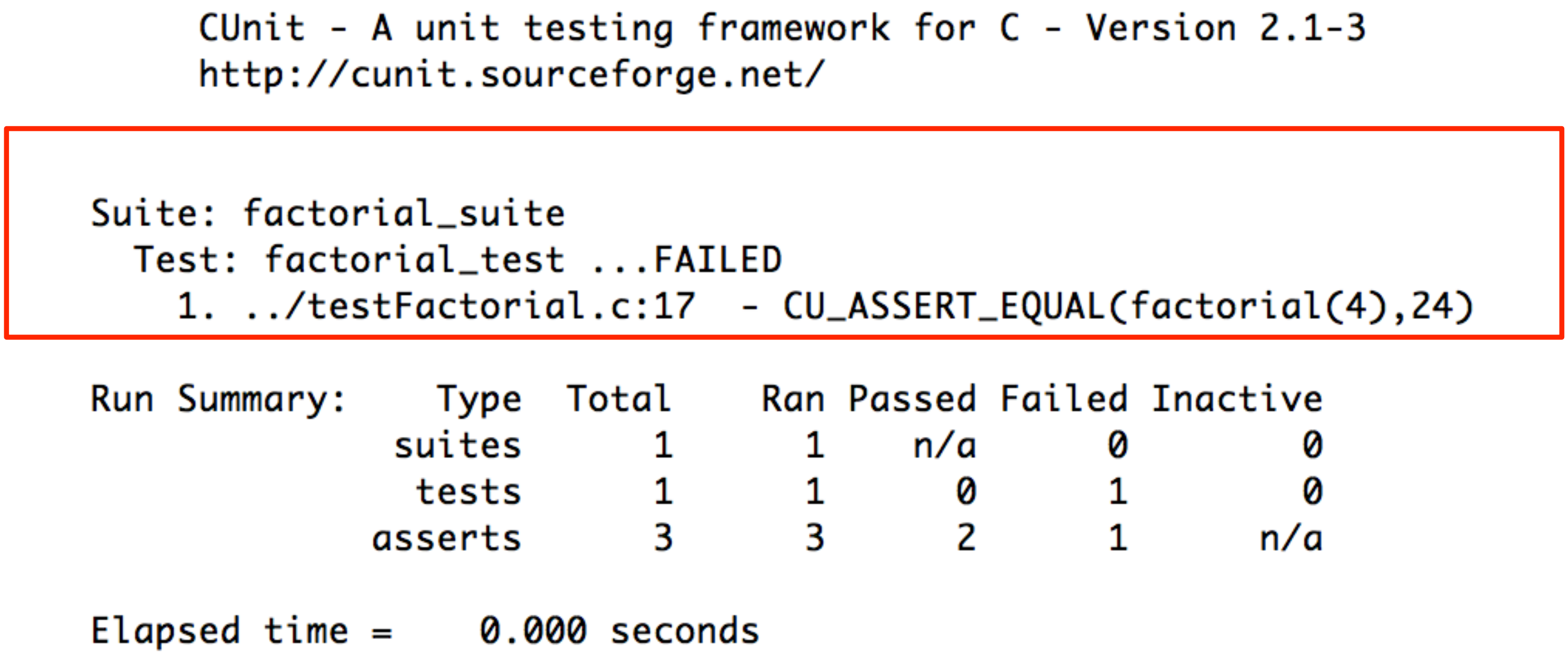
```
void factorial_testcase1(void){
    CU_ASSERT_EQUAL(factorial(0),1 );
    /* insert here 2 assertions necessary
     * to verify whether the factorial of 1 is 1
     * and whether the factorial of 4 is 24
     */
}
```

- Insert 2 additional assertions as prescribed and run the test case again.

# Design the Test Cases for the Factorial Function

```
void factorial_testcase1(void){
   CU_ASSERT_EQUAL(factorial(0),1 );
   CU_ASSERT_EQUAL(factorial(1),1 );
   CU_ASSERT_EQUAL(factorial(4),24 );
}
```

# One of the Assertions Fails

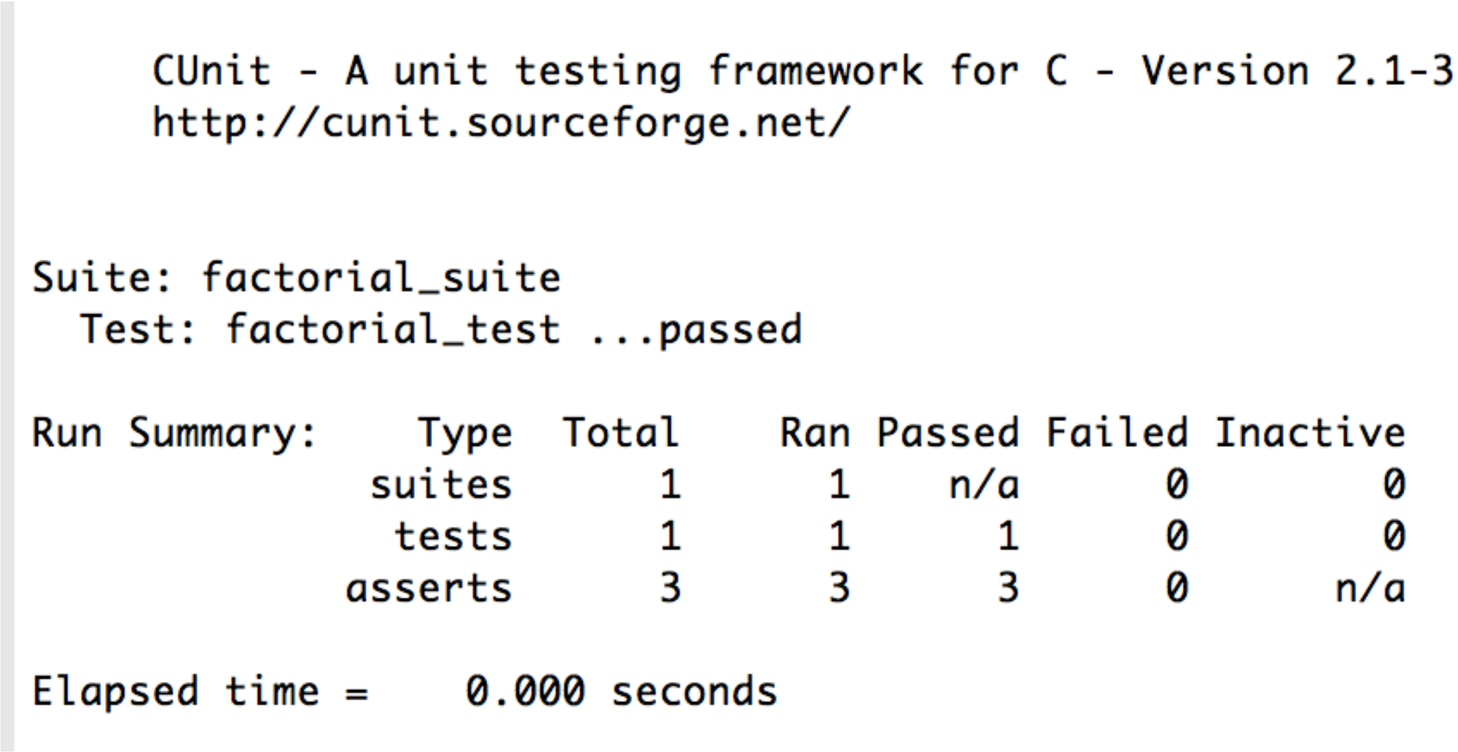


```
     CUnit - A unit testing framework for C - Version 2.1-3
     http://cunit.sourceforge.net/


Suite: factorial_suite
  Test: factorial_test ...FAILED
    1. ../testFactorial.c:17  - CU_ASSERT_EQUAL(factorial(4),24)

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests      1      1      0      1        0
             asserts      3      3      2      1      n/a

Elapsed time =    0.000 seconds
```

# Let’s Fix the Error inside factorial.c and Run the Test again

```
int factorial(int n) {
  //precondition: n >0
  int fact; //factorial of n
  int i; //to iterated between 1 and n
  /*calculates factorial of n */
  fact = 1;
  for(i=1; i<=n; i++) {
    fact *= i;
  }
  return fact;
}
```

# Now the Test is Successful



```
     CUnit - A unit testing framework for C - Version 2.1-3
     http://cunit.sourceforge.net/


Suite: factorial_suite
  Test: factorial_test ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests      1      1      1      0        0
             asserts      3      3      3      0      n/a

Elapsed time =    0.000 seconds
```