TESTOWANIE #1

CODERS SCHOOL

ŁUKASZ ZIOBROŃ

AGENDA

1. Framework GTest

TESTOWANIE

FRAMEWORK GTest

CODERS SCHOOL

DOKUMENTACJA GTESTA

- Repo GTest
- Primer
- Advanced Guide

PRZYDATNE ASERCJE

Fatal assertion	Nonfatal assertion	Verifies
`ASSERT_TRUE(condition);`	`EXPECT_TRUE(condition);`	condition is true
`ASSERT_FALSE(condition);`	`EXPECT_FALSE(condition);`	condition is false
`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
`ASSERT_THAT(value, matcher);`	`EXPECT_THAT(value, matcher);`	value matches matcher

ASERCJE Z WYJĄTKAMI

Fatal assertion	Nonfatal assertion	Verifies
`ASSERT_THROW(statement, exception_type);`	`EXPECT_THROW(statement, exception_type);`	statement throws an exception of the given type
`ASSERT_ANY_THROW(statement);`	`EXPECT_ANY_THROW(statement);`	statement throws an exception of any type
`ASSERT_NO_THROW(statement);`	`EXPECT_NO_THROW(statement);`	statement doesn't throw any exception

ORGANIZACJA TESTÓW

- `TEST(TestSuiteName, TestName)`
- `TEST_F(TestFixtureName, TestName)`

PRZYPADEK TESTOWY - `TEST`

```
TEST(ClassUnderTestSuite, callMeShouldAlwaysReturn42) {
    // GIVEN
    ClassUnderTest cut{};
    auto expected = 42;

    // WHEN
    auto result = cut.callMe();

    // THEN
    ASSERT_EQ(result, expected);
}
```

TESTY ZE WSPÓLNYMI DANYMI - TEST_F

```
struct ClassUnderTestFixture : public ::testing::Test {
    // common data and helper functions
    ClassUnderTest cut{};
}

TEST_F(ClassUnderTestFixture, callMeShouldAlwaysReturn42) {
    // GIVEN
    auto expected = 42;

    // WHEN
    auto result = cut.callMe();

    // THEN
    ASSERT_EQ(result, expected);
}
```

ZADANIE

Przetestuj algorytm `std::sort()`.

W tym zadaniu ważne jest pokrycie jak największej liczby scenariuszy tego algorytmu.

TESTY PARAMETRYCZNE - `TEST_P`

Potrzebujemy 3 rzeczy:

1. Klasę Fixture, dziedziczącą po `testing::TestWithParam<T>`, gdzie T jest typem danych wejściowych, które będą dostarczane do testu
2. Scenariusz testowy `TEST_P`
3. Generator `INSTANTIATE_TEST_SUITE_P`, który wygeneruje przypadki testowe

Link do dokumentacji

PRZYKŁAD

```
class MyFixture : public testing::TestWithParam<std::pair<int, int>> {
  // You can implement all the usual fixture class members here.
  // To access the test parameter, call GetParam() from class
  // TestWithParam<T>.
  ClassUnderTest cut;
};

TEST_P(MyFixture, MyTestName) {
  // Inside a test, access the test parameter with the GetParam() method
  // of the TestWithParam<T> class:
  auto [value, expected] = GetParam();
  EXPECT_EQ(cut.myFunction(value), expected);
}

INSTANTIATE_TEST_SUITE_P(MyInstantiationName,
                         MyFixture,
                         testing::Values({1, 1},
                                         {2, 1},
                                         {3, 2}));
```

GENERATORY

Dzięki generatorom danych jeden scenariusz testowy może zostać uruchomiony na różnych danych testowych.

- `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)`, `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 all combinations (Cartesian product) as `std::tuples` of the values generated by the N generators.

Link do dokumentacji

ZADANIE DODATKOWE

(Jeśli starczy czasu lub dla chętnych)

Wykonaj zadania z repo fan_controller. Będziemy na nim pracować na lekcji Testowanie #3. Oprócz polecenia z README.md dopisz też te same testy we frameworku GTest.

Repo fan_controller

TESTOWANIE

PRACA DOMOWA

CODERS SCHOOL

PRE-WORK

- Przeczytaj GMock for Dummies
- Zapoznaj się z testami w repozytorium Pizzas. Znajdziesz tam trochę kodu używającego mocków `TEST_F(PizzeriaTest, calculatePriceForPizzaMock)` oraz plik `test/mocks/PizzaMock.hpp`.
- Dopisz własne testy / mocki wedle uznania

BOWLING  - NOWY PROJEKT

Obejrzyj wideo Uncle Boba o TDD.

Do zrozumienia punktacji gry w kręgle przydatny może być ten opis zasad.

W nowych grupach napiszcie aplikację, która będzie zliczać punkty w kręgielni.

BOWLING 🎳

WYMAGANIA (+10 XP ZA KAŻDE SPEŁNIONE):

- liczenie punktów cząstkowych (dla niepełnych ramek, np: `3-|X|4/|5`)
- liczenie punktów całościowych - opis zasad
- walidacja inputu z niepełnymi ramkami dla kilku graczy (patrz następny slajd)
- input z wielu plików w jednym katalogu, każdy plik z kilkoma graczami reprezentuje inny tor (zalecane użycie Filesystem library z C++17)
- wyświetlanie wyników na ekranie z podziałem na tory (ze statusem gry) i graczy oraz zapis do jednego pliku (następny slajd)
- program (`main.cpp`) ma przyjmować 2 parametry z linii komend. Pierwszy to katalog, w którym będą pliki txt ze stanami gier na torach, a drugi opcjonalny to plik wyjściowy, w którym mają zostać zapisane przetworzone wyniki. Jeśli drugi parametr nie zostanie podany to wyniki mają zostać wypisane na ekran. Przykład użycia: `./bowling inputDirectory results.txt`. Program oczywiście ma działać i realizować powierzone zadanie.
- program (`main.cpp`) po podaniu parametru `-h` lub `--help` ma wyświetlać krótką informację o tym co robi i jak go używać (czyli punkt powyżej)

DOSTARCZENIE DO KOŃCA SIERPNIA +20 XP

BOWLING - INPUT

lane1.txt

```
Name1:X|4-|3
Name2:34|X|0-
:X|22|33
```

lane2.txt (pusty)

lane3.txt

```
Michael:X|7/|9-|X|-8|8/|-6|X|X|X||81
Radek:9-|9-|9-|9-|9-|9-|9-|9-|9-|9-||
```

BOWLING - OUTPUT

```
### Lane 1: game in progress ###
Name1 30
Name2 44
34
### Lane 2: no game ###
### Lane 3: game finished ###
Michael 167
Radek 90
```

WYMAGANIA ORGANIZACYJNE

- tablica dla projektu z podziałem na karteczki po planningu
- skonfigurowane Continuous Integration i system budowania
- praca przez pull requesty (każdy PR ma mieć nr i opis z karteczki, musi przejść wewnętrzne Code Review
- zawartość tablicy może się zmieniać w miarę odkrywania nowych wymagań (i na pewno to co założycie na początku się zmieni i dojdzie dużo rzeczy, których nie przewidzieliście)
- od samego początku spróbujcie pracować w trybie TDD
- każda funkcjonalność musi być przetestowana; brak testów = niespełnione wymaganie.
- pracujcie na forkach repo `coders-school/testing`
- po implementacji wszystkich wymagań PR do `coders-school/testing:master`

CODERS SCHOOL