C++ programok egységtesztelése googletest segítségével (GKxB_INTM006)

Dr. Hatwagner F. Miklós

Széchenyi István Egyetem, Győr

https://github.com/wajzy/GKxB_INTM006.git

2022. szeptember 25.

Tesztelés célja: a hibákat megtalálni üzembe helyezés előtt

Tesztelés alapelvei

1. A tesztelés bizonyos hibák jelenlétét jelezheti (ha nem jelzi, az nem jelent automatikusan hibamentességet)
2. Nem lehetséges kimerítő teszt (a hangsúly a magas kockázatú részeken van)
3. Korai teszt (minél hamarabb találjuk meg a hibát, annál olcsóbb javítani)
4. Hibák csoportosulása (azokra a modulokra/bemenetekre kell tesztelni, amelyre a legvalószínűbben hibás a szoftver)
5. Féregirtó paradoxon (a tesztesetek halmazát időnként bővíteni kell, mert ugyanazokkal a tesztekkel nem fedhetünk fel több hibát)
6. Körülmények (tesztelés alaposság függ a felhasználás helyétől, a rendelkezésre álló időtől, stb.)
7. A hibátlan rendszer téveszméje (A megrendelő elsősorban az igényeinek megfelelő szoftvert szeretne, és csak másodsorban hibamenteset; verifikáció vs. validáció)

Tesztelési technikák

Fekete dobozos (black-box, specifikáció alapú)

A tesztelő nem látja a forrást, de a specifikációt igen, és hozzáfér a futtatható szoftverhez. Összehasonlítjuk a bemenetekre adott kimeneteket az elvárt kimenetekkel.

Fehér dobozos (white-box, strukturális teszt)

Kész struktúrákat tesztelünk, pl.:

- kódsorok,
- elágazások,
- metódusok,
- osztályok,
- funkciók,
- modulok.

Lefedettség: a struktúra hány %-át tudjuk tesztelni a tesztesetekkel?

Egységteszt (unit test): a metódusok struktúra tesztje.

A tesztelés szintjei:

1. komponensteszt (egy komponens tesztelése)
   a. egységteszt
   b. modulteszt
2. integrációs teszt (kettő vagy több komponens együttműködése)
3. rendszerteszt (minden komponens együtt)
4. átvételi teszt (kész rendszer)

Kik végzik a tesztelést?

1-3 Fejlesztő cég

4 Felhasználók

Komponensteszt

- fehér dobozos teszt
- egységteszt
  - bemenet → kimenet vizsgálata
  - nem lehet mellékhatása
  - regressziós teszt: módosítással elronthattunk valamit, ami eddig jó volt → megismételt egységtesztek
- modulteszt
  - nem funkcionális tulajdonságok: sebesség, memóriaszivárgás (memory leak), szűk keresztmetszetek (bottleneck)

Integrációs teszt

- Komponensek közötti interfészek ellenőrzése, pl.
  - komponens - komponens (egy rendszer komponenseinek együttműködése)
  - rendszer - rendszer (pl. OS és a fejlesztett rendszer között)
- Jellemző hibaokok: komponenseket eltérő csapatok fejlesztik, elégtelen kommunikáció
- Kockázatok csökkentése: mielőbbi integrációs tesztekkel

Rendszerteszt: a termék megfelel-e a

- követelmény specifikációnak,
- funkcionális specifikációnak,
- rendszertervnek.

Gyakran fekete dobozos, külső cég végzi (elfogulatlanság)
Leendő futtatási környezet imitációja

Átvételi teszt, fajtái:

- alfa: kész termék tesztelése a fejlesztőnél, de nem általa (pl. segédprogramok)
- béta: szűk végfelhasználói csoport
- felhasználói átvételi teszt: minden felhasználó használja, de nem éles termelésben. Jellemző a környezetfüggő hibák megjelenése (pl. sebesség)
- üzemeltetői átvételi teszt: rendszergazdák végzik, biztonsági mentés, helyreállítás, stb. helyesen működnek-e

Rengeteg C++ egységteszt keretrendszerből lehet választani:

- Wiki oldal
- Exploring the C++ Unit Testing Framework Jungle
- C++ Unit Test Frameworks

Részletesen megvizsgáljuk: googletest

A googletest főbb tulajdonságai

- platformfüggetlen (Linux, Windows, Mac)
- független és megismételhető tesztek
- struktúrálható tesztek (teszt program $\rightarrow$ teszt csomag $\rightarrow$ teszteset)
- informatív
- leveszi a tesztelés technikai részének terhét a tesztelőről
- gyors (megosztott erőforrások)
- könnyen tanulható (xUnit architektúra)

- Fejlesztőkörnyezet (pl. VS Code, gcc) telepítés, ha még nincs
- cmake (forráskódok automatizált fordítása) telepítés (Ubuntu)
  `sudo apt install cmake` modern disztribúciókon
  Régi disztribúciókon, ahol a cmake verzió nem elég friss (3.14+)

Feladat

Készítsünk mátrixműveleteket megvalósító osztályt, ami elsőként egy mátrixszorzást valósít meg.

Az $A[a_{i,j}]_{m \times n}$ és $B[b_{i,j}]_{n \times p}$mátrixok szorzatán azt a $C[c_{i,j}]_{m \times p}$ mátrixot értjük, amelyre $c_{i,j} = a_{i,1} \cdot b_{1,j} + a_{i,2} \cdot b_{2,j} + \cdots + a_{i,n} \cdot b_{n,j} = \sum_{k=1}^{n} a_{i,k} \cdot b_{k,j}$

Megoldás

- Hozzuk létre a projekt könyvtárát: `mkdir 01`
- Lépjünk bele: `cd 01`
- Hozzuk létre a szükséges forráskódokat: `matrix01.h` (mátrix osztály), `example01.cpp` (főprogram)

01/matrix01.h

```
1  #include<vector>
2  #include<iostream>
3  namespace szeMatrix {
4
5  template<class T>
6  class Matrix {
7    protected:
8      std::vector<std::vector<T>> mtx;
9
10   public:
11     Matrix(std::vector<std::vector<T>> src) {
12       mtx = src;
13     }
```

01/matrix01.h

```
14      Matrix<T> mul(Matrix<T> right) const;
15      void print() const;
16      int getRowCount() const { return mtx.size(); }
17      int getColCount() const { return mtx[0].size(); }
18      T get(int row, int column) const { return mtx[row][column
            ↪ ]; }
19  };
```

01/matrix01.h

```
21 template<class T>
22 void Matrix<T>::print() const {
23   for(auto row : mtx) {
24     for(auto elem : row) {
25       std::cout << elem << '\t';
26     }
27     std::cout << std::endl;
28   }
29 }
```

01/matrix01.h

```
31 template<class T>
32 Matrix<T> Matrix<T>::mul(Matrix<T> right) const {
33   // Rows of left matrix and result matrix
34   int i = mtx.size();
35   // Columns of right matrix and res. matrix
36   int j = right.mtx[0].size();
37   // Columns of left matrix and rows of right matrix
38   int k = right.mtx.size();
39
40   // Creating an empty result matrix
41   std::vector<std::vector<T>> res;
42   // Resizing and filling it with zeros
43   res.resize(i, std::vector<T>(j, 0.));
```

01/matrix01.h

```
45    for(int r=0; r<i; r++) { // Matrix multiplication
46      for(int c=0; c<j; c++) {
47        for(int item=0; item<k; item++) {
48          res[r][c] += mtx[r][item]*right.mtx[item][c];
49        }
50      }
51    }
52
53    return Matrix(res);
54  }
55
56  }
```

01/example01.cpp

```
1  #include<vector>
2  #include"matrix01.h"
3
4  int main() {
5    std::vector<std::vector<int>> v1 = {
6      {11, 12, 13, 14},
7      {21, 22, 23, 24},
8      {31, 32, 33, 34}
9    };
10
11   std::vector<std::vector<int>> v2;
12   v2.resize(4, std::vector<int>(3, 1.));
```

01/example01.cpp

```
14    szeMatrix::Matrix<int> m1(v1);
15    szeMatrix::Matrix<int> m2(v2);
16    szeMatrix::Matrix<int> multiplied = m1.mul(m2);
17    multiplied.print();
18
19    return 0;
20  }
```

Fordítás (g++ -o example01 example01.cpp), kimenet:

```
50      50      50
90      90      90
130     130     130
```

Készítsünk az example01.cpp alapján googletest alapú tesztprogramot!

01/matrix01test.cpp

```
1  #include"matrix01.h"
2  #include<vector>
3  #include<gtest/gtest.h>
4
5  TEST(MulTest, meaningful) {
6    std::vector<std::vector<int>> left = {
7      {11, 12, 13, 14},
8      {21, 22, 23, 24},
9      {31, 32, 33, 34}
10   };
11   std::vector<std::vector<int>> right;
12   right.resize(4, std::vector<int>(3, 1.));
```

01/matrix01test.cpp

```
13    std::vector<std::vector<int>> expected = {
14      {50, 50, 50},
15      {90, 90, 90},
16      {130, 130, 130}
17    };
18    szeMatrix::Matrix<int> m1(left);
19    szeMatrix::Matrix<int> m2(right);
20    szeMatrix::Matrix<int> multiplied = m1.mul(m2);
```

01/matrix01test.cpp

```
21    ASSERT_EQ(expected.size(), multiplied.getRowCount());
22    ASSERT_EQ(expected[0].size(), multiplied.getColCount());
23    for(unsigned row=0; row<expected.size(); row++) {
24      for(unsigned col=0; col<expected[row].size(); col++) {
25        EXPECT_EQ(expected[row][col], multiplied.get(row, col));
26      }
27    }
28  }
```

A buildelést már a cmake végzi, hozzuk létre a konfigurációját!

01/CMakeLists.txt

```
1  cmake_minimum_required(VERSION 3.14)
2  project(01)
3
4  # GoogleTest requires at least C++14
5  set(CMAKE_CXX_STANDARD 14)
6
7  include(FetchContent)
8  FetchContent_Declare(
9    googletest
10   GIT_REPOSITORY https://github.com/google/googletest.git
11   GIT_TAG release-1.12.1
12 )
13 # For Windows: Prevent overriding the parent project's compiler/linker settings
14 set(gtest_force_shared_crt ON CACHE BOOL "" FORCE)
15 FetchContent_MakeAvailable(googletest)
```

01/CMakeLists.txt

```
17 enable_testing()
18
19 add_executable(
20   matrix_test
21   matrix01test.cpp
22 )
23 target_link_libraries(
24   matrix_test
25   GTest::gtest_main
26 )
27
28 include(GoogleTest)
29 gtest_discover_tests(matrix_test)
```

`cmake -S . -B build`
: Összeállító (build) környezet létrehozása.

`cmake -build build`
: Összeállítás indítása.

`cd build && ctest`
: Tesztprogram indítása.

Kimenet

```
Test project /home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/01/build
    Start 1: MulTest.meaningful
1/1 Test #1: MulTest.meaningful ...............   Passed    0.01 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   0.01 sec
```

Teszteset (test case)

"A set of preconditions, inputs, actions (where applicable), expected results and postconditions, developed based on test conditions."
(meaningful, ld. `matrix01test.cpp` 5. sor)

Tesztkészlet (test suite)

"A set of test cases or test procedures to be executed in a specific test cycle."
(MulTest, ld. `matrix01test.cpp` 5. sor)

Tesztprogram (test program)

Egy vagy több tesztkészletet foglal magába.

Sajnos a googletest nevezéktana következetlen:

googletest	**ISTQB**
teszt (test)	teszteset
teszteset (test case, test suite)	tesztkészlet

Assertion ($\approx$ állítás, követelés) Ellenőrizzük valamely elvárásunk teljesülését $\rightarrow$ siker (success), nem végzetes hiba (nonfatal failure), végzetes hiba (fatal failure). Makrók:

`EXPECT_*` nem végzetes hibát generál, ajánlott (több hiba jelezhető egyszerre)

`ASSERT_*` végzetes hibát generál, azonnal leállítja a tesztesetet (nincs értelme a folytatásnak; pl. ha két mátrix nem azonos méretű, nincs értelme az elemeiket összehasonlítgatni). Erőforrások felszabadítása, takarítás is elmarad!

Rontsuk el a kódot! („Elfelejtjük” összegezni a szorzatokat.)

02/matrix02.h (02/matrix02test.cpp, 02/CMakeLists.txt)

```
45    for(int r=0; r<i; r++) { // Matrix multiplication
46      for(int c=0; c<j; c++) {
47        for(int item=0; item<k; item++) {
48          // res[r][c] += mtx[r][item]*right.mtx[item][c];
49        }
50      }
51    }
```

Kimenet, LastTest.log

```
Test project /home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/02/build
    Start 1: MulTest.meaningful
1/1 Test #1: MulTest.meaningful ..............***Failed    0.00 sec

0% tests passed, 1 tests failed out of 1

Total Test time (real) =   0.01 sec

The following tests FAILED:
    1 - MulTest.meaningful (Failed)
Errors while running CTest
Output from these tests are in: /home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/02/build/Testing/Temporary/LastTest.log
Use "--rerun-failed --output-on-failure" to re-run the failed cases verbosely.
```

Most rontsuk el másképp a kódot! (Túl nagy lesz az eredmény mátrix.)

03/matrix03.h (03/CMakeLists.txt)

```
40    // Creating an empty result matrix
41    std::vector<std::vector<T>> res;
42    // Resizing and filling it with zeros
43    //res.resize(i, std::vector<T>(j, 0.));
44    res.resize(i*2, std::vector<T>(j, 0.));
```

03/matrix03test.cpp

```
21    ASSERT_EQ(expected.size(), multiplied.getRowCount())
22      << "A sorok szama elter! Elvart: " << expected.size()
23      << ", kapott: " << multiplied.getRowCount();
24    ASSERT_EQ(expected[0].size(), multiplied.getColCount())
25      << "Az oszlopok szama elter! Elvart: " << expected[0].size()
26      << ", kapott: " << multiplied.getColCount();
27    for(unsigned row=0; row<expected.size(); row++) {
28      for(unsigned col=0; col<expected[row].size(); col++) {
29        EXPECT_EQ(expected[row][col], multiplied.get(row, col))
30          << "Nem egyezik az elemek erteke a [" << row << "]["
31          << col << "] helyen!";
32      }
33    }
```

LastTest.log

```
Note: Google Test filter = MulTest.meaningful
[==========] Running 1 test from 1 test suite.
[----------] Global test environment set-up.
[----------] 1 test from MulTest
[ RUN      ] MulTest.meaningful
/home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/03/matrix03test.cpp:21: Failure
Expected equality of these values:
  expected.size()
    Which is: 3
  multiplied.getRowCount()
    Which is: 6
A sorok szama elter! Elvart: 3, kapott: 6
[  FAILED  ] MulTest.meaningful (0 ms)
[----------] 1 test from MulTest (0 ms total)

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

 1 FAILED TEST
```

- Az `ASSERT_EQ` leállította a tesztesetet.
- Testreszabott hibaüzeneteket jelenítettünk meg.

Elemi követelmények

Végzetes hibákhoz	**Nem végzetes hibákhoz**	**Követelmény**
ASSERT_TRUE(*feltétel*)	EXPECT_TRUE(*feltétel*)	*feltétel* igaz értékű
ASSERT_FALSE(*feltétel*)	EXPECT_FALSE(*feltétel*)	*feltétel* hamis értékű

Relációs követelmények

Végzetes hibákhoz	Nem végzetes hibákhoz	Követelmény
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*

Megjegyzések

- A feltüntetett operátoroknak definiáltnak kell lenniük *val1* és *val2* között. Lehetőségeink:
  1. Felültöltjük az operátorokat.
  2. Az {ASSERT,EXPECT}_{TRUE,FALSE} makrókat használjuk, de ezek nem írják a kimenetre az elvárt/kapott értékeket.
- A paraméterek egyszer lesznek kiértékelve, de nem definiált sorrendben (mellékhatások).
- Az {ASSERT,EXPECT}_EQ makrók mutatók esetén a címeket hasonlítja össze, nem az ott lévő tartalmat! C-stílusú karakterláncok kezeléséhez külön makrók léteznek. (`string` objektumokkal nincs gond.)
- C++11 szabványnak megfelelő fordító esetén `NULL` helyett `nullptr`-t használjunk (utóbbi nem konvertálható implicit módon `int`-té)!
- Lebegőpontos számok összehasonlításakor kerekítési hibák adódhatnak.

Készítsünk lebegőpontos számokból álló mátrixokat, majd teszteljük a szorzást ismét!

04/matrix04test.cpp (04/matrix04.h, 04/CMakeLists.txt)

```
31 TEST(MulTest, rounding) {
32   std::vector<std::vector<double>> left = {
33     {sqrt(2.), 0.},
34     {0., 1./3.}
35   };
36   std::vector<std::vector<double>> right;
37   right.resize(2, std::vector<double>(2, 1.));
38   std::vector<std::vector<double>> expected = {
39     {1.414213562, 1.414213562},
40     {0.333333333, 0.333333333}
41   };
```

04/matrix04test.cpp

```
42    szeMatrix::Matrix<double> m1(left);
43    szeMatrix::Matrix<double> m2(right);
44    szeMatrix::Matrix<double> multiplied = m1.mul(m2);
45    ASSERT_EQ(expected.size(), multiplied.getRowCount());
46    ASSERT_EQ(expected[0].size(), multiplied.getColCount());
47    for(unsigned row=0; row<expected.size(); row++) {
48      for(unsigned col=0; col<expected[row].size(); col++) {
49        EXPECT_EQ(expected[row][col], multiplied.get(row, col));
50      }
51    }
52  }
```

Kimenet

```
Test project /home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/04/build
    Start 1: MulTest.meaningful
1/2 Test #1: MulTest.meaningful ...............   Passed    0.00 sec
    Start 2: MulTest.rounding
2/2 Test #2: MulTest.rounding .................***Failed    0.00 sec

50% tests passed, 1 tests failed out of 2

Total Test time (real) =   0.00 sec

The following tests FAILED:
    2 - MulTest.rounding (Failed)
Errors while running CTest
Output from these tests are in: /home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/04/build/Testing/Temporary/LastTest.log
Use "--rerun-failed --output-on-failure" to re-run the failed cases verbosely.
```

LastTest.log

```
...
[ RUN      ] MulTest.rounding
/home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/04/matrix04test.cpp:49: Failure
Expected equality of these values:
  expected[row][col]
    Which is: 1.41421
  multiplied.get(row, col)
    Which is: 1.41421
...
/home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/04/matrix04test.cpp:49: Failure
Expected equality of these values:
  expected[row][col]
    Which is: 0.333333
  multiplied.get(row, col)
    Which is: 0.333333
...
```

A kerekítési hibák érzékelhetetlenek a kimeneten és a teszt sikertelen.

Próbálkozzunk a beépített, lebegőpontos számokat összehasonlító makrókkal!

05/matrix05test.cpp (05/matrix05.h, 05/CMakeLists.txt)

```
47    for(unsigned row=0; row<expected.size(); row++) {
48      for(unsigned col=0; col<expected[row].size(); col++) {
49        //EXPECT_EQ(expected[row][col], multiplied.get(row, col));
50        EXPECT_DOUBLE_EQ(expected[row][col], multiplied.get(row, col));
51      }
52    }
```

LastTest.log

```
[RUN      ] MulTest.rounding
/home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/05/matrix05test.cpp:50: Failure
Expected equality of these values:
  expected[row][col]
    Which is: 1.414213562
  multiplied.get(row, col)
    Which is: 1.4142135623730951
...
/home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/05/matrix05test.cpp:50: Failure
Expected equality of these values:
  expected[row][col]
    Which is: 0.33333333300000001
  multiplied.get(row, col)
    Which is: 0.33333333333333331
[  FAILED  ] MulTest.rounding (0 ms)
```

Most már látszik, hogy az értékek közötti különbség nagyobb, mint 4 ULP (Units in the Last Place), ezért tekinti őket a teszt különbözőnek.

Növeljük meg a számok közötti legnagyobb megengedett eltérést!

06/matrix06test.cpp (06/matrix06.h, 06/CMakeLists.txt)

```
47    for(unsigned row=0; row<expected.size(); row++) {
48      for(unsigned col=0; col<expected[row].size(); col++) {
49        //EXPECT_EQ(expected[row][col], multiplied.get(row, col));
50        //EXPECT_DOUBLE_EQ(expected[row][col], multiplied.get(row, col));
51        EXPECT_NEAR(expected[row][col], multiplied.get(row, col), 1e-9);
52      }
53    }
```

Kimenet

```
Test project /home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/06/build
    Start 1: MulTest.meaningful
1/2 Test #1: MulTest.meaningful ...............   Passed    0.00 sec
    Start 2: MulTest.rounding
2/2 Test #2: MulTest.rounding .................   Passed    0.00 sec

100% tests passed, 0 tests failed out of 2

Total Test time (real) =   0.00 sec
```

Lebegőpontos számokkal szemben támasztható követelmények

Végzetes hibákhoz	Nem végzetes hibákhoz	Követelmény
ASSERT_FLOAT_EQ(*val1*, *val2*);	EXPECT_FLOAT_EQ(*val1*, *val2*);	*float* típusú értékek 4 ULP-n belül
ASSERT_DOUBLE_EQ(*val1*, *val2*);	EXPECT_DOUBLE_EQ(*val1*, *val2*);	*double* típusú értékek 4 ULP-n belül
ASSERT_NEAR(*val1*, *val2*, *abs_error*);	EXPECT_NEAR(*val1*, *val2*, *abs_error*);	a két érték különbségének abszolút értéke nem nagyobb *abs_error*-nál

Próbáljuk meg a mátrixok elemenkénti összehasonlítása helyett a teljes mátrixokat összehasonlítani!

07/matrix07test.cpp (07/matrix07.h, 07/CMakeLists.txt)

```
31 TEST(MulTest, equality) {
32   std::vector<std::vector<double>> left = {
33     {11, 12, 13, 14},
34     {21, 22, 23, 24},
35     {31, 32, 33, 34}
36   };
37   std::vector<std::vector<double>> right;
38   right.resize(4, std::vector<double>(3, 1.));
39   std::vector<std::vector<double>> expected = {
40     {50, 50, 50},
41     {90, 90, 90},
42     {130, 130, 130}
43   };
```

07/matrix07test.cpp

```
44    szeMatrix::Matrix<double> m1(left);
45    szeMatrix::Matrix<double> m2(right);
46    szeMatrix::Matrix<double> mexp(expected);
47    szeMatrix::Matrix<double> multiplied = m1.mul(m2);
48    ASSERT_EQ(mexp.getRowCount(), multiplied.getRowCount());
49    ASSERT_EQ(mexp.getColCount(), multiplied.getColCount());
50    ASSERT_EQ(mexp, multiplied);
51  }
```

Kimenet

```
wajzy@wajzy-notebook:~/Dokumentumok/gknb_intm006/GKxB_INTM006/07$ cmake --build build
...
/home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/07/matrix07test.cpp:50:3:
  required from here
/home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/07/build/_deps/googletest-src/
  googletest/include/gtest/gtest.h:1358:11: error: no match for ‘operator==’
  (operand types are ‘const szeMatrix::Matrix<double>’ and
  ‘const szeMatrix::Matrix<double>’)
   if (lhs == rhs) {
       ~~~~^~~~~~
...
```

Probléma: az 50. sor `ASSERT_EQ(mexp, multiplied);` utasítása feltételezi az `==` operátor felültöltését a `Matrix` osztályhoz.

08/matrix08.h (08/matrix08test.cpp, 08/CMakeLists.txt)

```
 5  template<class T>
 6  class Matrix {

10    public:

19      template<class U>
20      friend bool operator==(const Matrix<U> &m1, const Matrix<U> &m2);
21  };

58  template<class U>
59  bool operator==(const Matrix<U> &m1, const Matrix<U> &m2) {
60    return m1.mtx==m2.mtx;
61  }
```

Kimenet

```
wajzy@wajzy-notebook:~/Dokumentumok/gknb_intm006/GKxB_INTM006/08$ cd build/ && ctest
Test project /home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/08/build
    Start 1: MulTest.meaningful
1/3 Test #1: MulTest.meaningful ...............   Passed    0.00 sec
    Start 2: MulTest.equality
2/3 Test #2: MulTest.equality .................   Passed    0.00 sec
    Start 3: MulTest.rounding
3/3 Test #3: MulTest.rounding .................   Passed    0.00 sec

100% tests passed, 0 tests failed out of 3

Total Test time (real) =   0.01 sec
```

Teszteljük le a `print()` tagfüggvény kimenetét!

Függvény	Funkció
`CaptureStdout()`	Megkezdi az `stdout`-ra írt tartalom rögzítését
`GetCapturedStdout()`	Lekérdezi a rögzített tartalmat és leállítja a rögzítést
`CaptureStderr()`	Megkezdi az `stderr`-re írt tartalom rögzítését
`GetCapturedStderr()`	Lekérdezi a rögzített tartalmat és leállítja a rögzítést

Belső tagfüggvények, használatuk nem javasolt (googletest forráskód).

09/matrix09.cpp (09/matrix09.h, 09/CMakeLists.txt)

```
76 TEST(MulTest, print) {
77   std::vector<std::vector<double>> right;
78   right.resize(2, std::vector<double>(2, 1.));
79   szeMatrix::Matrix<double> m2(right);
80   const char* expected = "1\t1\t\n1\t1\t\n";
81   testing::internal::CaptureStdout();
82   m2.print();
83   std::string output = testing::internal::GetCapturedStdout();
84   ASSERT_EQ(expected, output.c_str());
85 }
```

LastTest.log

```
...
[ RUN      ] MulTest.print
/home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/09/matrix09test.cpp:84: Failure
Expected equality of these values:
  expected
    Which is: 0x5576ac2320a2
  output.c_str()
    Which is: 0x7fff6fa4a800
[  FAILED  ] MulTest.print (0 ms)
...
```

Probléma: a C-stílusú karakterláncok címeit hasonlítja össze, nem az ott lévő tartalmat!

C-stílusú karakterláncokkal szemben támasztható követelmények

Végzetes hibákhoz	Nem végzetes hibákhoz	Követelmény
ASSERT_STREQ(*str1*, *str2*);	EXPECT_STREQ(*str1*, *str2*);	A két C-stílusú karakterlánc tartalma azonos
ASSERT_STRNE(*str1*, *str2*);	EXPECT_STRNE(*str1*, *str2*);	A két C-stílusú karakterlánc tartalma eltérő
ASSERT_STRCASEEQ(*str1*, *str2*);	EXPECT_STRCASEEQ(*str1*, *str2*);	A két C-stílusú karakterlánc tartalma a kis- és nagybetűk eltérésétől eltekintve azonos
ASSERT_STRCASENE(*str1*, *str2*);	EXPECT_STRCASENE(*str1*, *str2*);	A két C-stílusú karakterlánc tartalma a kis- és nagybetűk eltérését figyelmen kívül hagyva is eltérő

Javítsuk a tesztesetet és készítsünk további, hasonló tagfüggvényeket (tesztekkel)!

10/matrix10test.cpp (10/CMakeLists.txt)

```
76 TEST(MulTest, print) {
77   std::vector<std::vector<double>> right;
78   right.resize(2, std::vector<double>(2, 1.));
79   szeMatrix::Matrix<double> m2(right);
80   const char* expected = "1\t1\t\n1\t1\t\n";
81   testing::internal::CaptureStdout();
82   m2.print();
83   std::string output = testing::internal::GetCapturedStdout();
84   //ASSERT_EQ(expected, output.c_str());
85   ASSERT_STREQ(expected, output.c_str());
86 }
```

10/matrix10.h

```
3  #include<sstream>

7  class Matrix {

11   public:

16     void print() const;
17     std::string toString() const;
18     const char* toCString() const;

24 };
```

10/matrix10.h

```
36 template<class T>
37 std::string Matrix<T>::toString() const {
38   std::stringstream ss;
39   for(auto row : mtx) {
40     for(auto elem : row) {
41       ss << elem << '\t';
42     }
43     ss << std::endl;
44   }
45   return ss.str();
46 }
47
48 template<class T>
49 const char* Matrix<T>::toCString() const {
50   return toString().c_str();
51 }
```

10/matrix10test.cpp

```
88  TEST(MulTest, toString) {
89    std::vector<std::vector<double>> right;
90    right.resize(2, std::vector<double>(2, 1.));
91    szeMatrix::Matrix<double> m2(right);
92    std::string expected = "1\t1\t\n1\t1\t\n";
93    ASSERT_EQ(expected, m2.toString());
94  }
95
96  TEST(MulTest, toCString) {
97    std::vector<std::vector<double>> right;
98    right.resize(2, std::vector<double>(2, 1.));
99    szeMatrix::Matrix<double> m2(right);
100   const char* expected = "1\t1\t\n1\t1\t\n";
101   ASSERT_STREQ(expected, m2.toCString());
102 }
```

Vegyük észre, hogy a tesztünkben egyre többször ismétlődnek részek:

10/matrix10test.cpp

```
76  TEST(MulTest, print) {
77    std::vector<std::vector<double>> right;
78    right.resize(2, std::vector<double>(2, 1.));
79    szeMatrix::Matrix<double> m2(right);
80    const char* expected = "1\t1\t\n1\t1\t\n";


88  TEST(MulTest, toString) {
89    std::vector<std::vector<double>> right;
90    right.resize(2, std::vector<double>(2, 1.));
91    szeMatrix::Matrix<double> m2(right);
92    std::string expected = "1\t1\t\n1\t1\t\n";


96  TEST(MulTest, toCString) {
97    std::vector<std::vector<double>> right;
98    right.resize(2, std::vector<double>(2, 1.));
99    szeMatrix::Matrix<double> m2(right);
100   const char* expected = "1\t1\t\n1\t1\t\n";
```

Megoldás: teszt fixture-ök (≈alkatrész) használata

1. Származtassunk le egy osztályt a `::testing::Test`-ből! Ha az `Osztaly`-t szeretnénk tesztelni, legyen a neve `OsztalyTest`!
2. Deklaráljuk a többször használt tagokat! Legyenek védettek, hogy a leszármazottakból is használhatók legyenek!
3. A tagokat inicializáljuk az alapértelmezett konstruktorban vagy a (felüldefiniált) `SetUp()` tagfüggvényben!
4. Ha szükséges, készítsünk destruktort vagy (felüldefiniált) `TearDown()` tagfüggvényt az erőforrások felszabadítására!
5. Ha szükséges, írjunk függvényeket, amiket több teszteset is hívhat!

6. A tesztesetek definiálásakor a `TEST` helyett használjuk a `TEST_F` makrót!
7. A tesztkészlet neve egyezzen meg a fixture osztály nevével (`OsztalyTest`)!

Megjegyzések

- Az osztálynak már a tesztesetek makrói előtt definiáltnak kell lennie.
- Könnyű elgépelni a `SetUp()` és `TearDown()` függvények neveit, használjuk az `override` kulcsszót (C++11)!
- Minden egyes tesztesethez új példány készül a fixture-ből (nem „interferálnak” a tesztesetek), majd:
  alapértelmezett konstruktor $\rightarrow$ `SetUp()` $\rightarrow$ `TEST_F` $\rightarrow$ `TearDown()` $\rightarrow$ destruktor.

Mikor és miért érdemes konstruktort/destruktort használni?

- A `const` minősítővel ellátott tagváltozó csak a konstruktort követő inicializátor listával inicializálható. Jó ötlet a véletlen módosítások meggátolására.
- Ha a fixture osztályból származtatunk, az ős(ök) konstruktorának/destruktorának hívása mindenképpen végbemegy a megfelelő sorrendben. A `SetUp()`/`TearDown()` esetében erre a programozónak kell ügyelnie.

Mikor és miért érdemes a `SetUp()/TearDown()` függvényeket használni?

- A C++ nem engedi meg virtuális függvények hívását a konstruktorokban és destruktorokban, mert elvileg így meghívható lehetne egy inicializálatlan objektum metódusa, és ezt túl körülményes ellenőrizni. (Ha megengedi, akkor is csak az aktuális objektum metódusát hívja.)
- A konstruktorban/destruktorban nem használhatóak az ASSERT_* makrók. Megoldás:
  1. `SetUp()/TearDown()` használata
  2. Az egész tesztprogramot állítjuk le egy `abort()` hívással.
- Ha a leállási folyamat során kivételek kelethezhetnek, azt a destruktorban nem lehet megbízhatóan lekezelni (definiálatlan viselkedés, akár azonnali programleállással).

11/matrix11test.cpp (11/CMakeLists.txt, 11/matrix11.h)

```
6  class MatrixTest : public ::testing::Test {
7    protected:
8      szeMatrix::Matrix<double>* mtx2by2;
9      const char* expectedStr = "1\t1\t\n1\t1\t\n";
10     void SetUp() override {
11       std::vector<std::vector<double>> vec2by2;
12       vec2by2.resize(2, std::vector<double>(2, 1.));
13       mtx2by2 = new szeMatrix::Matrix<double>(vec2by2);
14     }
15     void TearDown() override {
16       delete mtx2by2;
17     }
18 };
```

11/matrix11test.cpp

```
90  TEST_F(MatrixTest, print) {
91    testing::internal::CaptureStdout();
92    mtx2by2->print();
93    std::string output = testing::internal::GetCapturedStdout();
94    ASSERT_STREQ(expectedStr, output.c_str());
95  }
96
97  TEST_F(MatrixTest, toString) {
98    std::string expected = expectedStr;
99    ASSERT_EQ(expected, mtx2by2->toString());
100 }
101
102 TEST_F(MatrixTest, toCString) {
103   ASSERT_STREQ(expectedStr, mtx2by2->toCString());
104 }
```

Kimenet

```
wajzy@lenovo:~/Dokumentumok/gknb_intm006/GKxB_INTM006/11$ cd build/ && ctest
Test project /home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/11/build
    Start 1: MulTest.meaningful
1/6 Test #1: MulTest.meaningful ...............   Passed    0.00 sec
    Start 2: MulTest.equality
2/6 Test #2: MulTest.equality .................   Passed    0.00 sec
    Start 3: MulTest.rounding
3/6 Test #3: MulTest.rounding .................   Passed    0.00 sec
    Start 4: MatrixTest.print
4/6 Test #4: MatrixTest.print .................   Passed    0.00 sec
    Start 5: MatrixTest.toString
5/6 Test #5: MatrixTest.toString ..............   Passed    0.00 sec
    Start 6: MatrixTest.toCString
6/6 Test #6: MatrixTest.toCString .............   Passed    0.00 sec

100% tests passed, 0 tests failed out of 6

Total Test time (real) =   0.01 sec
```

Egészítsük ki a `Matrix` osztályt olyan konstruktorral, ami egy `rows` sorból és `cols` oszlopból álló mátrixot véletlenszerűen feltölt `min` és `max` közé eső értékekkel!

12/matrix12.h (12/CMakeLists.txt)

```
 8 template<class T>
 9 class Matrix {

13   public:
14     Matrix(int rows, int cols, T min, T max);

27 };
```

12/matrix12.h

```
29 template<class T>
30 Matrix<T>::Matrix(int rows, int cols, T min, T max) {
31   unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
32   std::mt19937 rng(seed);
33   std::uniform_int_distribution<uint32_t> dist;
34   mtx.resize(rows, std::vector<T>(cols));
35   for(int r=0; r<rows; r++) {
36     for(int c=0; c<cols; c++) {
37       mtx[r][c] = 0.2 + min+(T)dist(rng)/rng.max()*(max-min); // BAD
38       // mtx[r][c] = min+(T)dist(rng)/rng.max()*(max-min); // GOOD
39     }
40   }
41 }
```

A BAD sor kizárólag tesztelési célokat szolgál, hogy néha intervallumon kívüli értékek kerüljenek a mátrixba.

12/matrix12test.cpp

```
90  TEST(MulTest, randomized) {
91    int rows = 2;
92    int cols = 3;
93    double min = -3.;
94    double max = +3.;
95    szeMatrix::Matrix<double> mtxRnd(rows, cols, min, max);
96    ASSERT_EQ(rows, mtxRnd.getRowCount());
97    ASSERT_EQ(cols, mtxRnd.getColCount());
98    for(int r=0; r<rows; r++) {
99      for(int c=0; c<cols; c++) {
100       double val = mtxRnd.get(r, c);
101       EXPECT_GE(max, val);
102       EXPECT_LE(min, val);
103     }
104   }
105 }
```

- Tesztkészlet `N`-szeri ismétlése. Negatív értékre az örökkévalóságig ismétel.
  `--gtest_repeat=N`
- Leállás az első olyan tesztkészlet iterációnál, ami hibát talált. Debuggerből futtatva a teszteket a memória tartalma ellenőrizhető.
  `--gtest_break_on_failure`
- Tesztesetek szűrése: csak akkor fut le egy teszteset, ha létezik olyan pozitív, de nem létezik olyan negatív minta, amire illeszkedik. A negatív minták elhagyhatóak. A pozitív mintákat a negatívaktól - választja el. A * tetszőleges karakterláncra illeszkedik, a ? egy tetszőleges karaktert helyettesít.
  `--gtest_filter=poz1:poz2:...:pozN-neg1:neg2:...:negN`
- Tesztkészletek és -esetek listázása
  `--gtest_list_tests`

Egyes beállítások környezeti változókon keresztül is módosíthatóak.

Milyen teszteseteink vannak?

```
wajzy@lenovo:~/Dokumentumok/gknb_intm006/GKxB_INTM006/12/build$ ./matrix_test --gtest_list_tests
Running main() from /home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/12/build/_deps/googletest-src/googletest/src/
  gtest_main.cc
MulTest.
  meaningful
  equality
  rounding
  randomized
MatrixTest.
  print
  toString
  toCString
```

- Minden tesztkészlet összes tesztesetének futtatása
  `./runTests`
  `./runTests --gtest_filter=*`
- Csak a `MulTest` tesztkészlet futtatása
  `./runTests --gtest_filter=MulTest.*`
- Az összes r betűt tartalmazó teszt futtatása, kivéve a String-et tartalmazókat és `MulTest.rounding`-ot, azaz `randomized` és `print` futtatása
  `./runTests --gtest_filter=*r*-*String:MulTest.rounding`
- Csak a `randomized` futtatása 100-szor
  `./runTests --gtest_filter=MulTest.randomized --gtest_repeat=100`

- Teszteredmények fájlba mentése. Tesztismétlés esetén csak az utolsó iteráció eredményét tartalmazza. Alapértelmezett kimenet: `test_detail.xml` Ha kimenet egy mappa, mindig új nevet választ a felülírás elkerülésére.
  `--gtest_output=xml<:kimenet>`
  Pl. `./runTests --gtest_filter=MulTest.randomized --gtest_output=xml:egysegteszt.xml`

12/build/egysegteszt.xml


```
−<testsuites tests="1" failures="1" disabled="0" errors="0" time="0" timestamp="2022-09-12T11:15:02.634" name="AllTests">
  −<testsuite name="MulTest" tests="1" failures="1" disabled="0" skipped="0" errors="0" time="0" timestamp="2022-09-12T11:15:02.634">
    −<testcase name="randomized" file="/home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/12/matrix12test.cpp" line="90" status="run"
     result="completed" time="0" timestamp="2022-09-12T11:15:02.634" classname="MulTest">
      −<failure message="/home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/12/matrix12test.cpp:101 Expected: (max) >= (val), actual: 3 vs
       3.06395" type="">
         /home/wajzy/Dokumentumok/gknb_intm006/GKxB_INTM006/12/matrix12test.cpp:101 Expected: (max) >= (val), actual: 3 vs 3.06395
       </failure>
     </testcase>
   </testsuite>
 </testsuites>
```


- Az XML megjeleníthető különféle eszközökkel, pl. Jenkins/xUnit-tal

Egészítsük ki a konstruktort kivétel dobásával, ha az eredeti vektor sorai nem azonos elemszámúak!

13/matrix13.h (13/CMakeLists.txt)

```
6  #include<stdexcept>
7  namespace szeMatrix {
8
9  template<class T>
10 class Matrix {
11   protected:
12     std::vector<std::vector<T>> mtx;
13
14   public:
15     Matrix(int rows, int cols, T min, T max);
16     Matrix(std::vector<std::vector<T>> src);

26 };
```

13/matrix13.h

```
41 template<class T>
42 Matrix<T>::Matrix(std::vector<std::vector<T>> src) {
43   bool firstRow = true;
44   unsigned numCols;
45   for(auto row : src) {
46     if(firstRow) {
47       numCols = row.size();
48       firstRow = false;
49     } else {
50       if(numCols != row.size()) {
51         throw std::range_error("Row lengths are different.");
52       }
53     }
54     mtx.push_back(row);
55   }
56 }
```

Módosítsuk és egészítsük ki tesztünket!

13/matrix13test.cpp

```
20 TEST(MulTest, meaningful) {
21   std::vector<std::vector<int>> left = {
22     {11, 12, 13, 14},
23     {21, 22, 23, 24},
24     {31, 32, 33, 34}
25   };
26   std::vector<std::vector<int>> right;
27   right.resize(4, std::vector<int>(3, 1.));
28   std::vector<std::vector<int>> expected = {
29     {50, 50, 50},
30     {90, 90, 90},
31     {130, 130, 130}
32   };
```

13/matrix13test.cpp

```
33    ASSERT_NO_THROW({
34      szeMatrix::Matrix<int> m1(left);
35      szeMatrix::Matrix<int> m2(right);
36      szeMatrix::Matrix<int> multiplied = m1.mul(m2);
37      ASSERT_EQ(expected.size(), multiplied.getRowCount());
38      ASSERT_EQ(expected[0].size(), multiplied.getColCount());
39      for(unsigned row=0; row<expected.size(); row++) {
40        for(unsigned col=0; col<expected[row].size(); col++) {
41          EXPECT_EQ(expected[row][col], multiplied.get(row, col));
42        }
43      }
44    });
45  }
```

13/matrix13test.cpp

```
47 TEST(MulTest, diffRowLengths) {
48   std::vector<std::vector<int>> invalid = {
49     {11},
50     {21, 22},
51     {31, 32, 33}
52   };
53   ASSERT_THROW(szeMatrix::Matrix<int> re(invalid),
54     std::range_error);
55 }
```

Kivételek kiváltásával szemben támasztható követelmények

Végzetes hibákhoz	Nem végzetes hibákhoz	Követelmény
ASSERT_THROW(*statement*, *exception_type*);	EXPECT_THROW(*statement*, *exception_type*);	*statement* hatására *exception_type* kivételnek kell keletkeznie
ASSERT_ANY_THROW(*statement*);	EXPECT_ANY_THROW(*statement*);	*statement* hatására valamilyen kivételnek kell keletkeznie
ASSERT_NO_THROW(*statement*);	EXPECT_NO_THROW(*statement*);	*statement* hatására semmilyen kivételnek sem szabad keletkeznie

A kivétel objektumban tárolt adatok (üzenet, egyedi hibakód, stb.) ezzel a módszerrel nem ellenőrizhetőek.

A haláltesztek (Death Tests) azt ellenőrzik, hogy valamilyen körülmény hatására a program leáll-e. Egészítsük ki a konstruktort úgy, hogy negatív sor- vagy oszlopszám esetén 1 hibakóddal álljon le a program!

14/matrix14.h (14/CMakeLists.txt)

```
28 template<class T>
29 Matrix<T>::Matrix(int rows, int cols, T min, T max) {
30   if(rows<0 or cols<0) {
31     std::cerr << "Row and column numbers must be non-negative.";
32     exit(1);
33   }
```

Ellenőrizzük, hogy a program valóban leáll-e az elvárt módon!

14/matrix14test.cpp

```
119 TEST(MatrixDeathTest, constructor) {
120   ASSERT_EXIT(szeMatrix::Matrix<double> mtxRnd(-1, 2, 1., 2.),
121               ::testing::ExitedWithCode(1),
122               "Row and column numbers must be non-negative.");
123 }
```

Haláltesztet támogató makrók

Végzetes hibákhoz	Nem végzetes hibákhoz	Követelmény
ASSERT_DEATH(*statement*, *matcher*);	EXPECT_DEATH(*statement*, *matcher*);	*statement* programleállást idéz elő *matcher* üzenettel
ASSERT_DEATH_IF_SUPPORTED(*statement*, *matcher*);	EXPECT_DEATH_IF_SUPPORTED(*statement*, *matcher*);	Csak akkor ellenőrzi, hogy *statement* programleállást idéz-e elő *matcher* üzenettel, ha a haláltesztek támogatottak
ASSERT_EXIT(*statement*, *predicate*, *matcher*);	EXPECT_EXIT(*statement*, *predicate*, *matcher*);	*statement* programleállást idéz elő *matcher* üzenettel, a kilépési kódot *predicate*-nek megfelelőre állítja

Paraméterezés:

statement

A programleálláshoz vezető (egyszerű vagy összetett) utasítás.

predicate

Függvény vagy függvény objektum, ami `int` paramétert vár és `bool`-t szolgáltat:

- `::testing::ExitedWithCode(exit_code)`
  Az elvárt kilépési kódot ellenőrzi.
- `::testing::KilledBySignal(signal_number)`
  Ellenőrzi, hogy a programot az elvárt jelzés szakította-e félbe (Windows-on nem támogatott).

Paraméterezés folyt.:

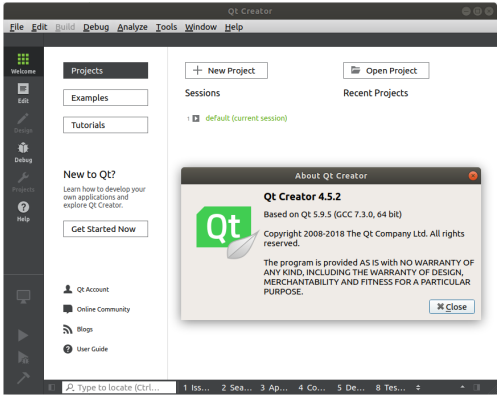
matcher

A szabvány hibacsatornára írt, elvárt üzenet. Ellenőrizhető: POSIX kiterjesztett reguláris kifejezéssel

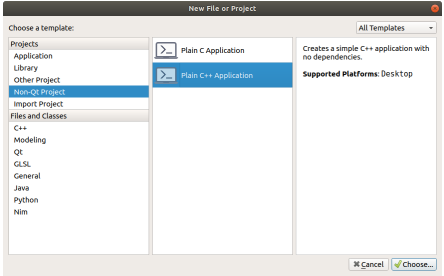
Megjegyzések

- A 0 kilépési kóddal leálló programot nem tekintik „halott” programnak. A leállítás általában `abort()`, `exit()` hívással vagy egy jelzéssel történik.
- A haláltesztek készletének neve `DeathTest`-re kell, hogy végződjön (részletek). Szálbiztos környezet szükséges lehet.

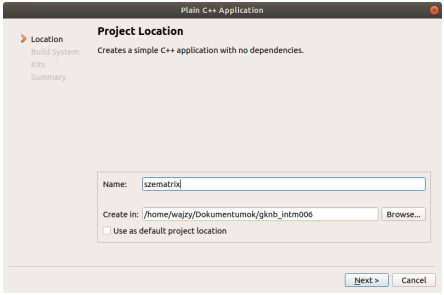
1. Indítsuk el a Qt Creatort.



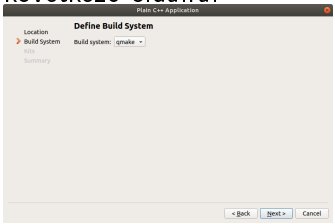
2. Készítsünk új projektet az osztályunk működésének kipróbálásához (egyelőre googletest nélkül)! *File* → *New File or Project...*
3. A dialógusablakban jelöljük meg a *Non-Qt Project*-et majd a *Plain C++ Application*-t! Végül kattintsunk a *Choose...* gombra!



4 A dialógusablakban adjuk meg a projekt nevét a *Name* mezőben (`szematrix`), és válasszunk egy mappát, ahová a projektet elhelyezhetjük! Végül lépjünk a következő oldalra (*Next*)!

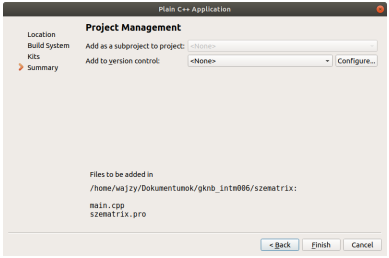


5. Az összeállító rendszer elvileg lehetne *cmake* is, de ezt az IDE nem támogatja teljeskörűen, ezért őrizzük meg az alapértelmezett *qmake*-et, majd lépjünk a következő oldalra!



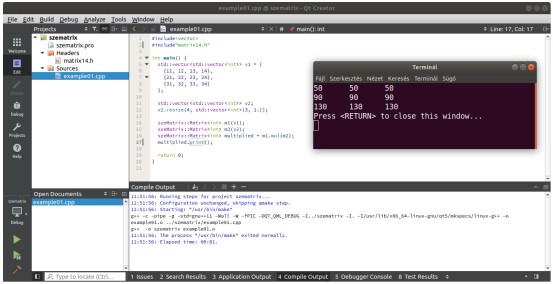
6. A készleteknél (*Kit Selection*) hagyjunk mindent változatlanul!

7 A következő oldalon, az *Add to version control...*-nál lehetne beállítani a Git verziókezelőt, de mivel nincs hosszabb távú célunk a projekttel, nem élünk a lehetőséggel.
A dialógusablak alján látszik, hogy létrejön a *qmake* projektleíró fájlja (`szematrix.pro`) illetve egy helykitöltő forrásszöveg (`main.cpp`), amit hamarosan lecserélünk.



8. Másoljuk a projekt mappájába a `01/example01.cpp` és `14/matrix14.h` fájlokat!
9. A *Projects* nézetben kattintsunk jobb gombbal a projekt nevén (*szematrix*), a kinyíló menüben pedig válasszuk az *Add Existing Files...* pontot! Adjuk hozzá a projekthez az előbb bemásolt két állományt!
10. Kattintsunk jobb gombbal a *main.cpp*-n, majd válasszuk a *Remove File...* lehetőséget, azaz töröljük a generált állományt! Ne sajnáljuk a fájlt véglegesen törölni (*Delete file permanently*).
11. Nyissuk meg az `example01.cpp` fájlt a nevére duplán kattintva, majd módosítsuk a második sort, hogy a `matrix14.h`-ra hivatkozzon!

12 Ezután a program fordítható, futtatható (*Build* → *Run*, vagy a zöld háromszögre kattintva).

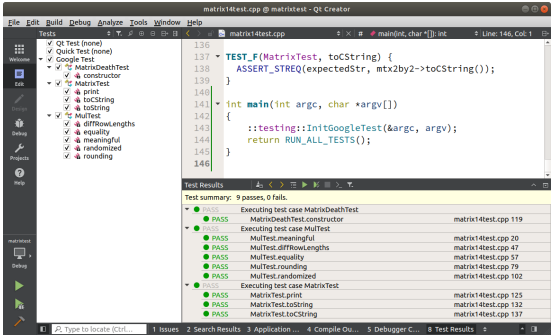


13. Készítsük elő a teszt projekt mappáját:
    1. `mkdir matrixtest`
    2. `cd matrixtest`
    3. `git clone https://github.com/google/googletest.git -b release-1.12.1`
    4. Másoljuk át a 14-es mappa `matrix14.h` és `matrix14test.cpp` fájljait a `matrixtest` könyvtárba!
14. A *File* → *New File or Project...* menüvel megnyíló ablakban most válasszuk az *Other Project*-et, majd az *Auto Test Project*-et!
15. A megnyíló dialógusablak *Name* mezőjébe gépeljük az új projekt nevét: `matrixtest`!

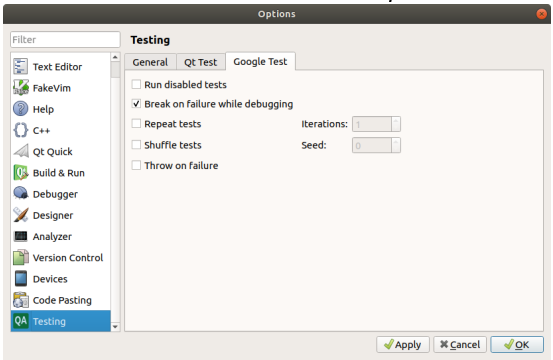
16 A következő fülön kell kiválasztani a tesztelés eszközét, ami legyen *Googletest*! Bár most erre semmi szükségünk, kénytelenek vagyunk megadni egy helykitöltő teszteset-nevet (*Test case name*, pl. case1) és tesztkészlet-nevet (*Test set name*, pl. set1). Engedélyezzük a C++11-kompatibilis fordítást a jelölőnégyzettel, majd adjuk meg a *googletest* könyvtárát! Őrizzük meg a *qmake* összeállító rendszert!

17. A *Next*, majd *Finish* gombokra kattintva létrejön, és aktívvá válik a teszt projekt.
18. A *Projects* nézetben a projekten jobb gombbal kattintva válasszuk az *Add Existing Files...* pontot, és adjuk hozzá a két átmásolt fájlt a projekthez!
19. Az automatikusan generált `main.cpp` main függvényét másoljuk a `matrix14test.cpp` fájl végére!
20. Távolítsuk el a projektből és töröljük a `main.cpp` és `tst_case1.h` fájlokat (Del gomb)!
21. A tesztprogram a zöld gombbal fordítható, futtatható.

- Kiválasztott/összes teszteset futtatása: *Test Results* kimeneti ablaktábla (*Window* → *Output Panes* → *Test Results*) zöld háromszögeivel



- További beállítások: *Tools* → *Options...* → *Testing*



Tesztelésről általában

Ficsor Lajos, Kovács László, Kusper Gábor, Krizsán Zoltán: Szoftvertesztelés

ISTQB CTFL Syllabus 2018

Szakkifejezések kereshető gyűjteménye

googletest

GoogleTest hivatalos oldal

Ubuntu-specifikus részletek

IBM tananyag a googletest-hez

További szakmai anyagok, érdeklődőknek

Tesztelési módszerek
Statikus kódellenőrzés
V&V általában