

Laboratorio de Programación - Labo07

Taller de Testing usando GTEST

Algoritmos y Estructuras de Datos I

Departamento de Computación, FCEyN, Universidad de
Buenos Aires.

Qué es un caso de prueba?

Damos una entrada válida y verificamos que la salida sea la esperada.

```
1    ...
2    vector<int> minimoAlFinal = {1,4,6,8,2,0};
3    string nomTest = "Minimo al final"
4    printBeginTest(1, nomTest);
5    bool res = asegurar(indiceMinSubsec(minimoAlFinal, 0,5) == 5, true);
6    if(res)
7        cout << "Correcto." << endl;
8    else {
9        cout << "Incorrecto. ";
10       cout << "Esperado: 5. ";
11       cout << "Calculado: " << indiceMinSubsec(minimoAlFinal, 0,5);
12       cout << endl;
13   }
14   printEndTest(1, nomTest);
15   ...
```

Por qué testear?

- ▶ Para verificar que el programa cumple con una especificación (formal o informal).
- ▶ Porque no siempre se puede demostrar la correctitud de un sistema en un tiempo razonable.
- ▶ Porque el programa debe ser robusto a un funcionamiento en producción.
- ▶ Económica: los costos para reparar sistemas que salieron a producción y ya están en el cliente... quién los paga?

Testear es suficiente para encontrar todos los errores? Meta: si los inputs están razonablemente acotados sí. En la vida real no.

Por qué testear?

Según el tipo de test:

- ▶ Test de unidad (unit test): sirve para testear una funcionalidad particular.
- ▶ Test de integración: para testear la interacción entre una funcionalidad nueva y el resto del sistema.
- ▶ Test de sistema: para probar el comportamiento de un sistema con varias partes.
- ▶ y muchas otras...

Qué testear?

TODO.



Qué testear?

Quisiéramos cubrir todas las entradas posibles para el problema.

Pero cuando el problema tiene muchas (incluso infinitas) entradas posibles, no se puede probar todo.



Tests de caja negra

También llamados test funcionales o basados en la especificación. Son los tests en los que NO se conoce la implementación de los programas que se prueban. Se cuenta, por otro lado, con el resultado esperado por el sistema a una entrada dada (por especificación).

Hay que elegir casos de prueba que sean *representativos* del universo de casos posibles.

Cómo consigo un conjunto de casos representativo?

- ▶ Conjuntos representativos en el que los elementos sean parecidos los unos a los otros.
- ▶ *Particionar* el universo y tomar representantes de cada parte.

Tests de caja negra

Ejemplos de particiones dependiendo el problema:

- ▶ Por valor absoluto de la diferencia entre dos números
- ▶ Por divisibilidad, primo o y no primos.
- ▶ Casos borde.

Tests de caja blanca

Se los llama tests estructurales. Son los tests en los que SÍ se conoce la implementación de los programas que se prueban.

La estrategia para estos casos es intentar cubrir todos los flujos de código posibles con el conjunto de los casos. Si el test suite no llegó a cubrir alguna parte del programa, se busca incluir tests adicionales para que esta parte sea verificada.

Tests de caja blanca

Un juego de azar consiste en sacar bolitas y sumar puntos según la siguiente regla:

- ▶ Si la cantidad de bolitas es menor que 10, gana dos puntos por cada bolita que sacó. Si no, un punto por cada una.
- ▶ Además, si la cantidad de bolitas que sacó es múltiplo de 3, gana 10 puntos. Si no, pierde 10 puntos.

Cuántos puntos ganó Sofía?

Tests de caja blanca

```
1  int puntaje(int b) {
2      int res;
3      if (b < 10) {
4          res = 2 * b;
5      } else {
6          res = b;
7      }
8      if (b % 3 == 0) {
9          res = res + 10;
10     } else {
11         res = res - 10;
12     }
13     return res;
14 }
```

Tests de caja blanca

Si hago un test con b=9

```
1  int puntaje(int b) {  
2      int res; // pasa por aca  
3      if (b < 10) { // pasa por aca  
4          res = 2 * b; // pasa por aca  
5      } else {  
6          res = b;  
7      }  
8      if (b % 3 == 0) { // pasa por aca  
9          res = res + 10; // pasa por aca  
10     } else {  
11         res = res - 10;  
12     }  
13     return res; // pasa por aca  
14 }
```

que otros valores de b puedo usar para que pase por las otras líneas de código?

Google Test

Google Test nos va ayudar a definir nuestros propios test

En este taller se provee código para utilizar Google Test en CLion.

- ▶ En el template-alumnos.zip van a encontrar:
 - ▶ **lib** - Directorio conteniendo los fuentes del google test (obtenidos de <https://github.com/google/googletest>). Hay que descomprimirlo
 - ▶ **main.cpp y CMakeList.txt** - Ambos archivos estan inicializados para ejecutar los tests del ejercicio 1.
 - ▶ Ejercicios para testear, empezando por **esPrimo.cpp, esPrimo.h y Test/esPrimoTEST.cpp**

Google Test

La primera función a testear está en ej1/esPrimo.cpp:

```
1  #include "esPrimo.h"
2
3  bool esPrimo(int n) {
4      if (n < 2) {
5          return false;
6      } else {
7          for (int i = 2; i < n; i++) {
8              if (n % i == 0) {
9                  return false;
10             }
11         }
12         return true;
13     }
14 }
```

Google Test

En ej1/Test/esPrimoTest.cpp, ya están definidos tres tests:

```
1  #include "gtest/gtest.h"
2  #include "../esPrimo.h"
3
4  TEST(EsPrimoTest, NumeroPrimo) {
5      // setup
6      int n = 7;
7      //exercise
8      bool result = esPrimo(n);
9      // check
10     EXPECT_TRUE(result);
11 }
```

Google Test

▶ Estructura

- ▶ Cada test por separado, con la macro TEST
- ▶ El test tiene un nombre y un grupo

▶ Expectations / Assertions

- ▶ Assert() : ASSERT_* y EXPECT_*
- ▶ * puede ser: TRUE, FALSE, EQ, LT, STREQ, etc
- ▶ EXPECT_* : NO FATAL
- ▶ ASSERT_* : FATAL (Interrumpe el test)

Google Test

Retomando el caso del test de ejemplo, esPrimo, la verificación puede escribirse:

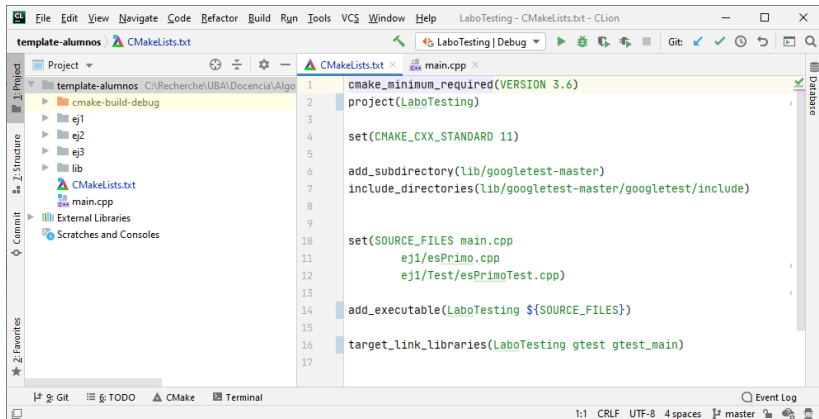
```
1      // check
2      EXPECT_TRUE(result);
3      o EXPECT_EQ(true,result);
4      o ASSERT_TRUE(result);
5      o ASSERT_EQ(true,result);
```

Retomando el caso del test de ejemplo, esPrimo, la verificación puede escribirse:

```
1 // * {ASSERT|EXPECT}_EQ(v1, v2): Test v1 == v2
2 // * {ASSERT|EXPECT}_NE(v1, v2): Test v1 != v2
3 // * {ASSERT|EXPECT}_LT(v1, v2): Test v1 < v2
4 // * {ASSERT|EXPECT}_LE(v1, v2): Test v1 <= v2
5 // * {ASSERT|EXPECT}_GT(v1, v2): Test v1 > v2
6 // * {ASSERT|EXPECT}_GE(v1, v2): Test v1 >= v2
```

Google Test en CLion

Al cargar el proyecto, en la ventana de inicio vamos a ver:



El CMakeList.txt vincula solo los archivos fuente y test del problema esPrimo del ejercicio 1. Para compilar con los otros ejercicios, es preciso incluir cada fuente con su test entre los paréntesis del `set(SOURCE_FILES...`

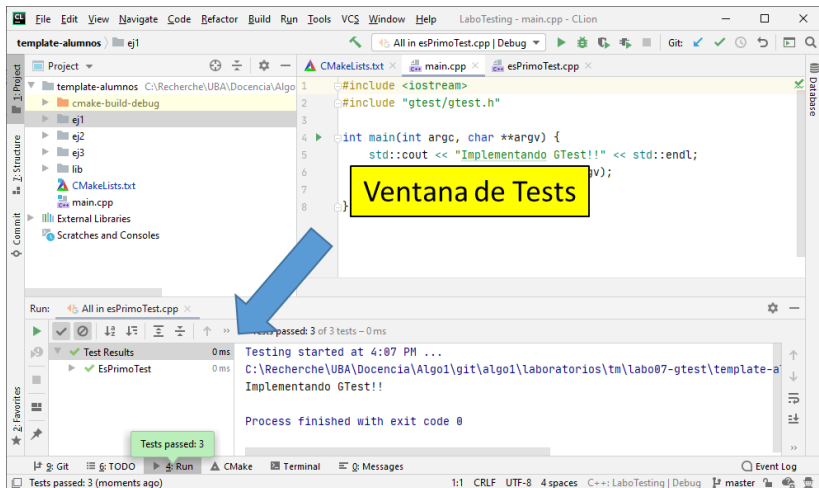
Google Test en CLion

En main.cpp:

```
1 #include <iostream>
2 #include "gtest/gtest.h"
3
4 int main(int argc, char **argv) {
5     std::cout << "Implementando GTest!!" << std::endl;
6     testing::InitGoogleTest(&argc, argv);
7     return RUN_ALL_TESTS();
8 }
```

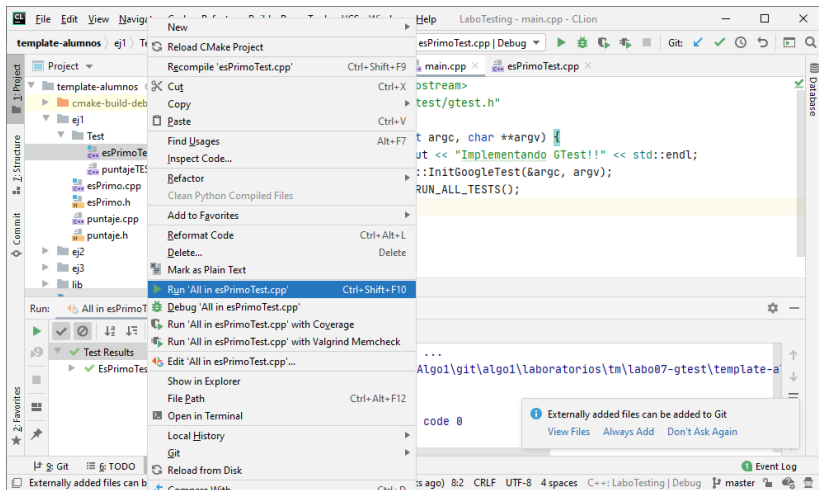
Google Test en CLion

Al correr el proyecto, se abre un box especial que muestra los resultados de los tests. En este caso, los 3 test pasaron con éxito.



Google Test en CLion

También podemos correr solo los test de `esPrimoTest.cpp` (botón derecho "Run All in `esPrimoTest.cpp`")



Bibliografía

- ▶ M. Pezzè, M. Young. Software Testing and Analysis: Process, Principles, and Techniques.
 - ▶ Capítulo 9: Test Case Selection and Adequacy
 - ▶ Capítulo 10: Functional Testing
 - ▶ Capítulo 12: Structural Testing