Software Testing

Overview

- What is testing and why is it important
- Testing Framework
  - CUnit
- Examples

Testing can only show the presence of errors and not their absence

E. Dijkstra

Difficulties of Testing

- Perception by some developers and managers:
  - Testing is seen as a novice's job.
  - Assigned to the least experienced team members.
  - Done as an afterthought (if at all).
    - *"My code is good; it won't have bugs. I don't need to test it."*
    - *"I'll just find the bugs by running the client program."*

- Limitations of what testing can show you:
  - It is impossible to completely test a system.
  - Testing does not always directly reveal the actual bugs in the code.
  - Testing does not prove the absence of errors in software.

Three Stages of Testing

- **Development Testing:**
  - System is tested during development to discover bugs and defects
- **Release Testing:**
  - A separate team tests a complete version of the software system before its release
  - Check that the system meets the stakeholders' requirements
- **User Testing (or Beta Testing):**
  - Users test the system in their own environment
  - Decide whether the software should be accepted or further development is required

Three Stages of Testing

- **Development Testing:**
  - System is tested during development to discover bugs and defects
- **Release Testing:**
  - A separate team tests a complete version of the software system before its release
  - Check that the system meets the stakeholders' requirements
- **User Testing (or Beta Testing)**
  - Users test the system in their own environment
  - Decide whether the software should be accepted or further development is required

Development Testing

- Aimed to discover bugs in the software
  - Interleaved with **debugging**
- Carried out by the software programmers
- Three levels of granularity:
  - Unit testing
    - Individual program units are tested.
    - Testing functionality of methods
  - Component testing
    - Testing component interfaces
  - System testing
    - Some or all the components in a system are integrated and the system is tested as a whole
    - Testing components interactions

Development Testing

- Aimed to discover bugs in the software
  - Interleaved with **debugging**
- Carried out by the software programmers
- Three levels of granularity:
  - Unit testing
    - Individual program units are tested.
    - Testing functionality of methods
  - Component testing
    - Testing component interfaces
  - System testing
    - Some or all the components in a system are integrated and the system is tested as a whole
    - Testing components interactions

Automation

Tests are encoded in a program that is run each time the system under development is to be tested

Very important for <u>regression testing</u>!

Automating Test Cases Execution

Unit-Testing Framework

- A unit-testing framework is a software tool for writing and running unit tests

- Provides reusable test functionality which:
  - Is easier to use
  - Is standardized
  - Enables automatic execution (important for regressions tests)

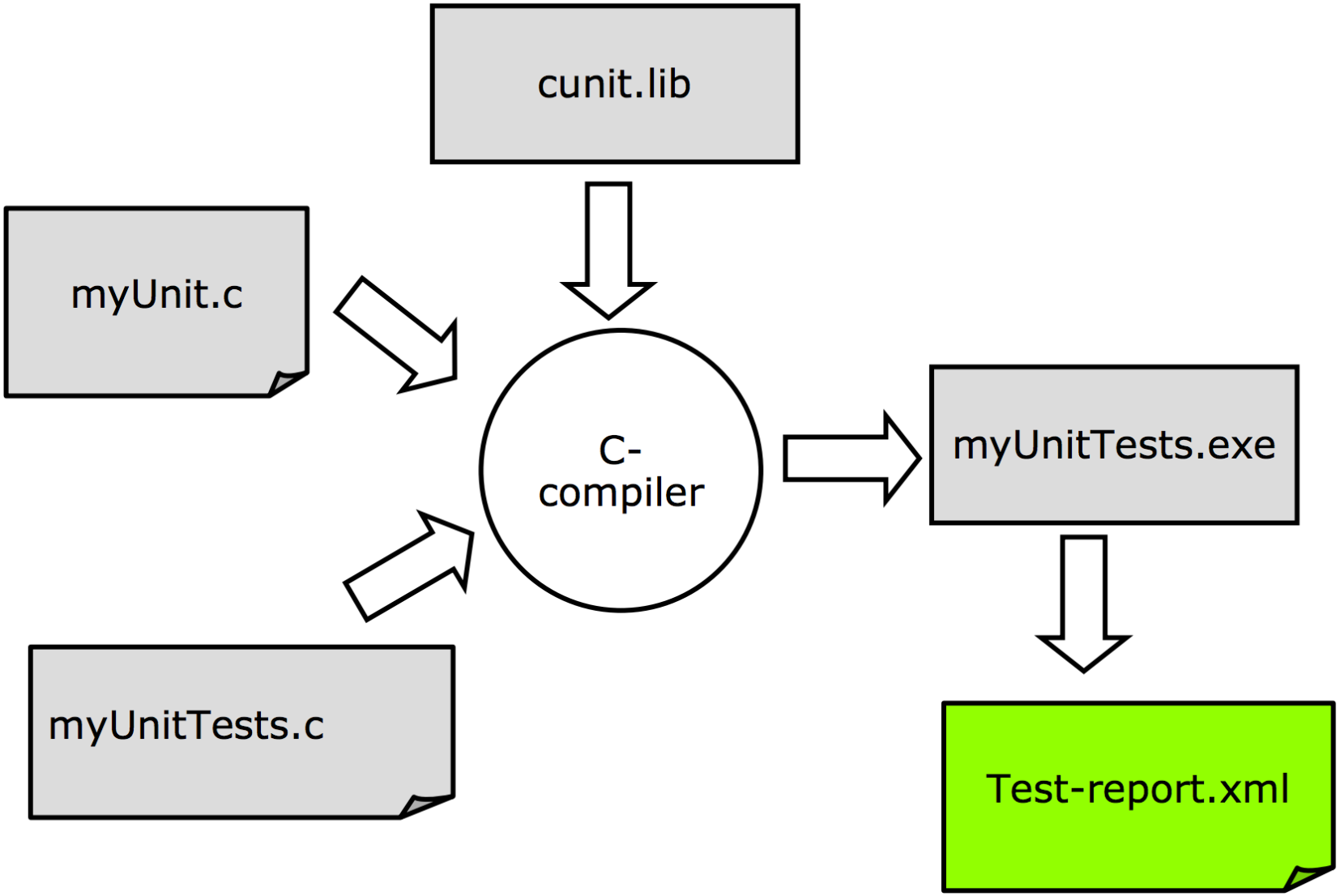
CUnit (https://sourceforge.net/projects/cunit/)

Why Unit-Testing Framework

- Lightweight tool that uses the same language and development environment as the programmer
- Offers an easy, systematic, and comprehensive way of organizing and executing tests:
  - It is practical to collect and re-use test cases
- Automatic Regression Testing
- Test report generation

CUnit

Basic Use of CUnit Framework



Module to be tested: `maxFunction.c`

maxFunction.h

```
int maxi(int i1, int i2);
```

maxFunction.c

```
#include "maxFunction.h"

int maxi(int i1, int i2){
        if (i1 > i2)
                return i1;
        else return i2;
}
```

Returns the greatest between 2 integers i1 and i2

Main Concepts in a Unit Test

- **Assertions**
  - Boolean expressions that compare expected and actual results
  - The basic and smallest building-block
- **Test Case**
  - A composition of concrete test procedures
  - May contain several assertions and test for several test objectives
  - E.g., all test of a particular function
- **Test Suite**
  - Collection of related test cases
  - Can be executed automatically in a single command

CUnit Assertions

CUnit Assertion	Meaning
CU_ASSERT(int *expression*)	Checks that expression is `TRUE` (non-zero)
CU_ASSERT_TRUE(*value*)	Checks that value is `TRUE` (non-zero)
CU_ASSERT_FALSE(*value*)	Checks that value is `FALSE` (zero)
CU_ASSERT_EQUAL(*actual, expected*)	Checks that *actual == expected*
CU_ASSERT_NOT_EQUAL(*actual, expected*)	Checks that *actual != expected*
CU_ASSERT_PTR_EQUAL(*actual, expected*)	Checks that pointers *actual == expected*
CU_ASSERT_PTR_NOT_EQUAL(*actual, expected*)	Checks that pointers *actual != expected*
CU_ASSERT_PTR_NOT_NULL(*value*)	Checks that pointer *value != NULL*
CU_ASSERT_DOUBLE_EQUAL(*actual, expected*)	Checks that doubles *actual* and *expected* are equivalent
CU_ASSERT_DOUBLE_NOT_EQUAL(*actual, expected*)	Checks that doubles *actual* and *expected* are not equivalent

http://cunit.sourceforge.net/doc/writing_tests.html

Module Implementing the tests: `test.c`

```
#include <stdio.h>
#include <stdlib.h>
#include "maxFunction.h"
#include <CUnit/CUnit.h>
#include <CUnit/Basic.h>
void test_maxi(void){
    CU_ASSERT(maxi(0,2) == 2);
    CU_ASSERT(maxi(0,-2) == 9);
    CU_ASSERT(maxi(1,2) == 2);
}
int main() {
    CU_initialize_registry();
    CU_pSuite suite = CU_add_suite("maxi_test", 0, 0);
    CU_add_test(suite, "maxi_fun", test_maxi);
    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    CU_cleanup_registry();
    return 0;
}
```

Necessary modules for creating test cases using cunit

test.c

```
#include <stdio.h>
#include <stdlib.h>
#include "maxFunction.h"
#include <CUnit/CUnit.h>
#include <CUnit/Basic.h>
void test_maxi(void){
    CU_ASSERT(maxi(0,2) == 2);
    CU_ASSERT(maxi(0,-2) == 9);
    CU_ASSERT(maxi(1,2) == 2);
}
int main() {
    CU_initialize_registry();
    CU_pSuite suite = CU_add_suite("maxi_test", 0, 0);
    CU_add_test(suite, "maxi_fun", test_maxi);
    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    CU_cleanup_registry();
    return 0;
}
```

Test case

test.c

- The test registry is the repository for suites and associated tests.
- CUnit maintains an active test registry which is updated when the user adds a suite or test.
- The user always has to initialize the registry before use and cleanup afterwards.

```
}
int main() {
    CU_initialize_registry();
    CU_pSuite suite = CU_add_suite("maxi_test", 0, 0);
    CU_add_test(suite, "maxi_fun", test_maxi);
    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    CU_cleanup_registry();
    return 0;
}
```

test.c

- Creates a new test collection (suite) having a specified name, initialization function and cleanup function.
- The suite's name must be unique among all suites in the registry.
- The initialization and cleanup functions are optional and are invoked respectively before and after the test contained in the suite are executed.

```
int main() {
    CU_initialize_registry();
    CU_pSuite suite = CU_add_suite("maxi_test", 0, 0);
    CU_add_test(suite, "maxi_fun", test_maxi);
    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    CU_cleanup_registry();
    return 0;
}
```

test.c

- Creates a new test having a specified name (*maxi_fun*) and test function (*test_maxi*) and registers it with the specified suite.
- The suite must have already been created.
- The test's name must be unique among all tests added to a single suite.
- Test functions have neither arguments nor return values.

```
    CU_initialize_registry();
    CU_pSuite suite = CU_add_suite("maxi_test", 0, 0);
    CU_add_test(suite, "maxi_fun", test_maxi);
    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    CU_cleanup_registry();
    return 0;
}
```

test.c

```
#include <stdio.h>
#include <stdlib.h>
```

- Set the run mode for the basic interface:
- Options are:
  - CU_BRM_NORMAL
  - CU_BRM_SILENT
  - CU_BRM_VERBOSE

```
    CU_pSuite suite = CU_add_suite(                    , 0, 0);
    CU_add_test(suite, "maxi_fun", tes    axi);
    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    CU_cleanup_registry();
    return 0;
}
```

test.c

```
#include <stdio.h>
#include <stdlib.h>
#include "maxFunction.h"
#include <CUnit/CUnit.h>
#include <CUnit/Basic.h>

void test_maxi(void){
    CU_ASSERT(maxi(0,2) == 2);
    CU_ASSERT(maxi(0,-2) == 9);
    CU_AS
}

int main
    CU_i
    CU_p

    CU_add_test(suite, "maxi_fun", test_

    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    CU_cleanup_registry();

    return 0;
}
```

- Run all CUnit tests registered in the registry using the basic interface

test.c

```
#include <stdio.h>
#include <stdlib.h>
#include "maxFunction.h"
#include <CUnit/CUnit.h>
#include <CUnit/Basic.h>

void test_maxi(void){
    CU_ASSERT(maxi(0,2) == 2);
    CU_ASSERT(maxi(0,-2) == 9);
    CU_ASSERT(maxi(1,2) == 2);
}

int main() {
    CU_initialize_registry(
    CU_pSuite suite = CU_ad

    CU_add_test(suite, "max

    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    CU_cleanup_registry();

    return 0;
}
```

- Cleans up the registry

Test Report

- Obtained after executing `MyTestingProject`

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


Suite: maxi_test
  Test: maxi_fun ...FAILED
    1. /Users/liliana 1/Documents/Work/Teaching/COMP10050/Lecture8/MyTestingProject/test.c:19  - maxi(0,-2) == 9

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 returned 0 (0x0)   execution time : 0.010 s
Press ENTER to continue.
```

Test Report

- Obtained after executing `MyTestingProject`

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


Suite: maxi_test
  Test: maxi_fun ...FAILED
    1. /Users/liliana 1/Documents/Work/Teaching/COMP10050/Lecture8/MyTestingProject/test.c:19  - maxi(0,-2) == 9

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 returned 0 (0x0)   execution time : 0.010 s
Press ENTER to continue.
```

Example 2: triangle.c

```
char * checkTriangle(int a, int b, int c){
   if(a == 90 || b == 90 || c == 90)
      return "Right";

   if(a == 60 && b == 60 && c == 60)
      return "Equilateral";

   if(a == b || b == c || c == a)
      return "Isosceles";

   return "Scalene";

}
```

Example 2: test.c

```
void triangle_testcase1(void){
    CU_ASSERT(strcmp("Equilateral", checkTriangle(60,60,60)) == 0);
    CU_ASSERT_STRING_EQUAL("Right", checkTriangle(40,90,50));
    CU_ASSERT_STRING_NOT_EQUAL("Isosceles", checkTriangle(80,80,50));
}

void runAllTests(){
    CU_initialize_registry();
    CU_pSuite suite = CU_add_suite("triangle_suite", 0, 0);
    CU_add_test(suite, "triangle_test", triangle_testcase1);
    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    CU_cleanup_registry();
}

int main() {
    runAllTests();
    return 0;
}
```

Test Report

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

Suite: triangle_suite
  Test: triangle_test ...FAILED
    1. ../test.c:18  - 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
```

Example 3

```
// calculate the slope of the line
double getSlope(double x1, double y1, double x2, double y2 ) {
    // avoid dividing by zero
    if(x1 == x2){
        perror("It is impossible to calculate the slope of a vertical line");
        return -1;
    }
    else return (y2 - y1) / (x2 - x1);
}
```

```
// calculate the perimeter of the line
double getDistance(double x1, double y1, double x2, double y2) {
    return sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1));
}
```

Cases

Case 1: the line is horizontal

Case 2: the line is neither horizontal nor vertical

Case 3: the line vertical

Cases

Case 1: the line is horizontal

Check that returned slope is 0

Case 2: the line is neither horizontal nor vertical

Case 3: the line vertical

Cases

Case 1: the line is horizontal

Check that returned slope is 0

Case 2: the line is neither horizontal nor vertical

- **Check that returned slope is > 0**
- **Check that returned slope is < 0**

Case 3: the line vertical

Check that returned slope is equal to - 1

In practice

Case 1: the line is horizontal

Case 2: the line is neither horizontal nor vertical

Case 3: the line vertical

In practice

Case 1: the line is horizontal

```
void slopeZero_testcase() {
        CU_ASSERT_EQUAL(getSlope(0,5,90,5),0); }
```

Case 2: the line is neither horizontal nor vertical

Case 3: the line vertical

In practice

Case 1: the line is horizontal

```
void slopeZero_testcase() {
        CU_ASSERT_EQUAL(getSlope(0,5,90,5),0); }
```

Case 2: the line is neither horizontal nor vertical

```
void slopeMiddle_testcase() {
        CU_ASSERT(getSlope(0,0,90,54) > 0);
        CU_ASSERT(getSlope(0,90,4,54) < 0); }
```

Case 3: the line vertical

In practice

Case 1: the line is horizontal

```
void slopeZero_testcase() {
        CU_ASSERT_EQUAL(getSlope(0,5,90,5),0); }
```

Case 2: the line is neither horizontal nor vertical

```
void slopeMiddle_testcase() {
        CU_ASSERT(getSlope(0,0,90,54) > 0);
        CU_ASSERT(getSlope(0,90,4,54) < 0); }
```

Case 3: the line vertical

```
void slopeError_testcase() {
        CU_ASSERT(getSlope(0,90,0,54) == -1); }
```

How to run the test cases

```
void runAllTests(){
    CU_initialize_registry();
    CU_pSuite suite = CU_add_suite("slope_suite", 0, 0);

    CU_add_test(suite, "slopeZero_test", slopeZero_testcase);
    CU_add_test(suite, "slopeMiddle_test", slopeMiddle_testcase);
    CU_add_test(suite, "slopeError_test", slopeError_testcase);

    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    CU_cleanup_registry(); }

int main() {
   runAllTests();
   return 0; }
```

How to run the test cases

```
void runAllTests(){
    CU_initialize_registry();
    CU_pSuite suite = CU_add_suite("slope_suite", 0, 0);

    CU_add_test(suite, "slopeZero_test", slopeZero_testcase);
    CU_add_test(suite, "slopeMiddle_test", slopeMiddle_testcase);
    CU_add_test(suite, "slopeError_test", slopeError_testcase);

    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    CU_cleanup_registry(); }

int main() {
   runAllTests();
   return 0; }
```

How to run the test cases

```
void runAllTests(){
    CU_initialize_registry();
    CU_pSuite suite = CU_add_suite("slope_suite", 0, 0);

    CU_add_test(suite, "slopeZero_test", slopeZero_testcase);
    CU_add_test(suite, "slopeMiddle_test", slopeMiddle_testcase);
    CU_add_test(suite, "slopeError_test", slopeError_testcase);

    CU_basic_set_mode(CU_BRM_VERBOSE);
    CU_basic_run_tests();
    CU_cleanup_registry(); }

int main() {
   runAllTests();
   return 0; }
```

Test Report

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


Suite: slope_suite
  Test: slopeZero_test ...passed
  Test: slopeMiddle_test ...passed
  Test: slopeError_test ...passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests      3      3      3      0        0
             asserts      4      4      4      0      n/a

Elapsed time =    0.000 seconds
It is impossible to calculate the slope of a vertical line: Undefined error: 0
```

Conclusion

- Code that isn’t tested doesn’t work
- Code that isn’t regression tested breaks eventually
- A unit testing framework enables efficient and effective unit and regression testing
- Use CUnit to store and maintain all the small tests that you write anyway
- Write tests instead of playing with printf is better because tests can be repeated automatically