
Faculty of Engineering
Cairo University

# Final Report Submission

## Software Testing

Submitted to:
Dr. Ahmed Sobeih
Eng. Ahmed Bahgat
Eng. Mohamed Adel

Submitted by:

| | |
|---|---|
| Name: Ahmed Saleh | Email: abosalaah96@gmail.com |
| Name: Mohamed Hamada | Email: homshamada@gmail.com |
| Name: Mina Magdy | Email: mina_mego5@yahoo.com |
| Name: Omar Osama | Email: omarosamasobeih@yahoo.com |

# Table of Contents

# Tool Overview:

We used tool named “CppUTest” and it is a C /C++ based unit xUnit test framework for unit testing and for test-driving your code. It is written in C++ but is used in C and C++ projects and frequently used in embedded systems but it works for any C/C++ project.

And the tool has many assertions like:

1. CHECK_EQUAL(expected, actual) - checks for equality between entities using “==” operator. So, to use it we need to have this operator in the datatype we are comparing.

2. STRCMP_EQUAL(expected, actual) - compares two “ char * “ arrays using “strcmp” function

3. POINTERS_EQUAL(expected, actual) - compares two pointers and check if they are equal and equal here means they have the same address.

4. CHECK_THROWS(expected_exception, expression) - checks if expression throws expected_exception (e.g. std::exception). CHECK_THROWS is only available if CppUTest is built with the Standard C++ Library (default).

5. FAIL(text) - always fails

# How to build it

- Windows Visual Studio
    - Create Visual Studio Project
    - Project Properties -> C/C++ -> General ->
    - Additional Include Directories -> path of include folder in the library folder
    - Project Properties -> Linker -> General -> Additional Library Directories -> path of lib folder in the library folder
    - Project Properties -> Linker -> Input -> Additional Dependencies -> Add “CppUTestd.lib”

- Linux : using this command
    - apt-get install cpputest

- MacOSX : using this command
    - brew install cpputest

# Logic Coverage:

1. First example: testing math function indicates happy numbers
   Happy number in his definition that has 2 or more features (even, has perfect square root or lucky i.e. all digits are either ‘4’ or ‘7’).

```
if ((hasSqrt && isEven) || (isLucky && (hasSqrt || isEven))) return true;
else return false;
```

| # | isLucky | isEven | hasSqrt | P | P(isLucky) | P(isEven) | P(hasSqrt) | Input |
|---|---|---|---|---|---|---|---|---|
| 1 | True | True | True | True | False | False | False | 4 |
| 2 | True | True | False | True | True | True | False | 74 |
| 3 | True | False | True | True | True | False | True | - |
| 4 | True | False | False | False | False | True | True | 7 |
| 5 | False | True | True | True | False | True | True | 16 |
| 6 | False | True | False | False | True | False | True | 2 |
| 7 | False | False | True | False | True | True | False | 81 |
| 8 | False | False | False | False | False | False | False | 3 |

```
TEST_GROUP(Math_Happy) {

};

TEST(Math_Happy, Test1) {
    CHECK(checkHappy(74));
}

TEST(Math_Happy, Test2) {
    CHECK_FALSE(checkHappy(7));
}

TEST(Math_Happy, Test3) {
    CHECK(checkHappy(16));
}

TEST(Math_Happy, Test4) {
    CHECK_FALSE(checkHappy(2));
}
```

i. Combinatorial Clause Coverage (COC):
Infeasible to force the combination where "isLucky" is true, "isEven" is false and "hasSqrt" is true

ii. Restricted Active Clause Coverage (RACC):

- Test Pair that satisfy RACC with respect to clause "isLucky": (2,6)
- Test Pair that satisfy RACC with respect to clause "isEven": (4,5)
- Test Pair that satisfy RACC with respect to clause "hasSqrt": (5,6)

iii. Restricted Inactive Clause Coverage (RICC):
Infeasible to force predicate to evaluate with true while "isEven" is major

iv. Correlated Active Clause Coverage (CACC):

- Test Pair that satisfy CACC with respect to clause "isLucky": (2,6)
- Test Pair that satisfy CACC with respect to clause "isEven": (4,5)
- Test Pair that satisfy CACC with respect to clause "hasSqrt": (5,6)

v. General Active Clause Coverage (GACC):

- Test Pair that satisfy GACC with respect to clause "isLucky": (2,6)
- Test Pair that satisfy GACC with respect to clause "isEven": (4,5)
- Test Pair that satisfy GACC with respect to clause "hasSqrt": (5,6)

vi. General Inactive Clause Coverage (GICC):
Infeasible to force predicate to evaluate with true while "isEven" is major

vii. Clause Coverage (CC):

- Test Pair that satisfy CC with respect to clause "isLucky": (1, 8)
- Test Pair that satisfy CC with respect to clause "isEven": (1, 8)
- Test Pair that satisfy CC with respect to clause "hasSqrt": (1, 8)

viii. Predicate Coverage (PC): Test Pair that satisfy predicate coverage: (5, 6)

2. Second Example: testing operator[] in multiset implemented using Fenwick tree

```
if (idx < 1 || idx > cnt) {
    throw out_of_range("ERROR :: trying to access an out of range element");
}
```

| # | idx < 1 | idx > cnt | P | P(idx<1) | P(idx>cnt) | idx |
|---|---|---|---|---|---|---|
| 1 | True | True | True | False | False | - |
| 2 | True | False | True | True | False | 0 |
| 3 | False | True | True | False | True | cnt + 1 |
| 4 | False | False | False | True | True | 1 |

```
TEST_GROUP(FenwickTree_Operator) {
    fenwick_multiset fm;
    int n;
    void setup() {
        n = MAX_N;
        for (int i = 0; i < n; ++i)
            fm.insert(rand() % MAX_V + 1);
    }
    void teardown() {

    }
};

TEST(FenwickTree_Operator, Test1) {
    CHECK_THROWS(out_of_range, fm[0]);
}
```

```
TEST(FenwickTree_Operator, Test2) {
    int ret;
    try {
        ret = fm[1];
    }
    catch (exception e) {
        FAIL("test fail");
        return;
    }
    LONGS_EQUAL(ret, fm[1]);
}


TEST(FenwickTree_Operator, Test3) {
    CHECK_THROWS(out_of_range, fm[n + 1]
}
```

i. Combinatorial Clause Coverage (COC):

Infeasible requirement

ii. Restricted Active Clause Coverage (RACC):

- Test Pair that satisfy RACC with respect to clause “idx < 1”: (2,4)
- Test Pair that satisfy RACC with respect to clause “idx > cnt”: (3, 4)

iii. Restricted Inactive Clause Coverage (RICC):

Infeasible requirement

iv. Clause Coverage (CC):

- Test Pair that satisfy CC with respect to clause “idx < 1”: (2,3)
- Test Pair that satisfy CC with respect to clause “idx > cnt”: (2, 3)

v. Predicate Coverage (PC):

Test Pair that satisfy predicate coverage: (3, 4)

3. Third Example: testing insert function in multiset implemented using Fenwick tree.

```
if (val < 1 || val > N) {
    throw exception("ERROR :: invalid value to insert");
}
```

| # | val < 1 | val > N | P | P(val<1) | P(val>N) | val |
|---|---|---|---|---|---|---|
| 1 | True | True | True | False | False | - |
| 2 | True | False | True | True | False | 0 |
| 3 | False | True | True | False | True | N + 1 |
| 4 | False | False | False | True | True | 1 |

```
TEST_GROUP(FenwickTree_Insert) {
    fenwick_multiset fm;
    int n;
    void setup() {
        n = MAX_N;
        for (int i = 0; i < n; ++i)
            fm.insert(rand() % MAX_V + 1);
    }
    void teardown() {

    }
};

TEST(FenwickTree_Insert, Test1) {
    CHECK_THROWS(out_of_range, fm[0]);
}
```

```
TEST(FenwickTree_Insert, Test2) {
    int ret;
    try {
        ret = fm[1];
    }
    catch (exception e) {
        FAIL("test fail");
        return;
    }
    LONGS_EQUAL(ret, fm[1]);
}

TEST(FenwickTree_Insert, Test3) {
    CHECK_THROWS(out_of_range, fm[n + 1]);
}
```

i. Combinatorial Clause Coverage (COC):

Infeasible requirement

ii. Restricted Active Clause Coverage (RACC):

- Test Pair that satisfy RACC with respect to clause “val < 1”: (2,4)
- Test Pair that satisfy RACC with respect to clause “val > N”: (3, 4)

iii. Restricted Inactive Clause Coverage (RICC):

Infeasible requirement

iv. Clause Coverage (CC):

- Test Pair that satisfy CC with respect to clause “val < 1”: (2,3)
- Test Pair that satisfy CC with respect to clause “val > N”: (2, 3)

v. Predicate Coverage (PC):

Test Pair that satisfy predicate coverage: (2, 4)

# Other Coverage Criteria:

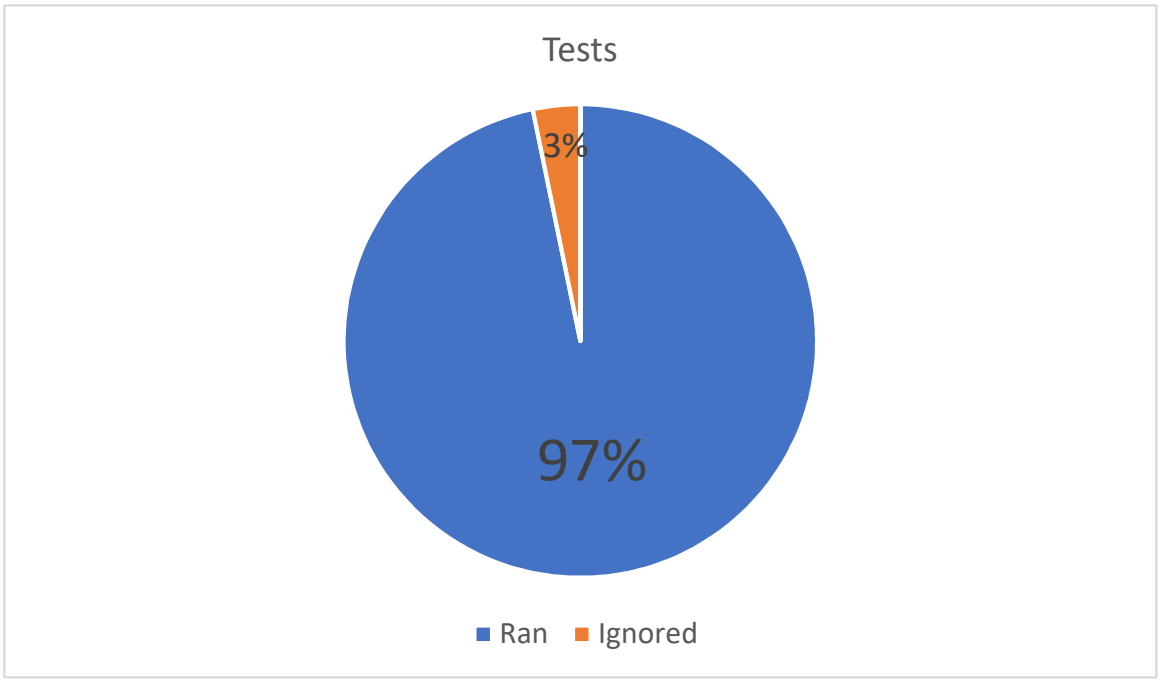
```
TEST(MathUtilsTestGroup, Test1)
{
    CHECK_EQUAL(3, gcd(6, 9));
    CHECK_EQUAL(1, gcd(10000, 23));
    CHECK_EQUAL(100, gcd(10000, 100));
    CHECK_EQUAL(13, gcd(13, 143));
    CHECK_EQUAL(13, gcd(26, 143));
}



TEST(MathUtilsTestGroup, Test2)
{
    LONGS_EQUAL(18, lcm(6, 9));
    LONGS_EQUAL(230000, lcm(10000, 23));
    LONGS_EQUAL(10000, lcm(10000, 100));
    LONGS_EQUAL(143, lcm(13, 143));
    LONGS_EQUAL(286, lcm(26, 143));
}

TEST(MathUtilsTestGroup, Test3)
{
    LONGS_EQUAL(9, powerr(3, 2));
    LONGS_EQUAL(25, powerr(5, 2));
    LONGS_EQUAL(216, powerr(6, 3));
    LONGS_EQUAL(100000, powerr(10, 5));
    LONGS_EQUAL(121, powerr(11, 2));
}
```

```
TEST(MathUtilsTestGroup, Test6)
{
    CHECK_FALSE(isPrime(10));
    CHECK_FALSE(isPrime(15));
    CHECK_FALSE(isPrime(18));
    CHECK_FALSE(isPrime(49));
    CHECK_FALSE(isPrime(56));
    CHECK_FALSE(isPrime(1000));
    CHECK_FALSE(isPrime(1234));
    CHECK_FALSE(isPrime(1e9));
}

TEST(MathUtilsTestGroup, Test7)
{
    DOUBLES_EQUAL(12.56637061, circleArea(2), 0.000001);
    DOUBLES_EQUAL(530.9291585, circleArea(13), 0.1);
    DOUBLES_EQUAL(31415.92654, circleArea(100), 0.1);
    DOUBLES_EQUAL(314159265.4, circleArea(10000), 0.1);
}

IGNORE_TEST(MathUtilsTestGroup, Test8)
{
    DOUBLES_EQUAL(12.56637061, circleArea(2), 0.000001);
    DOUBLES_EQUAL(530.9291585, circleArea(13), 0.000001);
    DOUBLES_EQUAL(31415.92654, circleArea(100), 0.0000000001);
    DOUBLES_EQUAL(314159265.4, circleArea(10000), 0.000001);
}
```

```
TEST(MathUtilsTestGroup, Test10)
{
    vector<int> expectedDivisors;
    int arr[] = { 1,2,3,4,6,9,12,18,36 };
    for (int i = 0; i < 9; i++) {
        expectedDivisors.push_back(arr[i]);
    }
    vector<int> divs = getDivisors(36);
    CHECK(expectedDivisors.size() == divs.size());
    for (int i = 0; i < divs.size(); i++)
    {
        LONGS_EQUAL(expectedDivisors[i], divs[i]);
    }
}
```

# Statistics:

Checks = 102
Tests = 62
60 Tests ran
2 Tests ignored
0 Tests filtered out



# References:

https://cpputest.github.io/
https://cpputest.github.io/manual.html

# Work Load:

| Name | Task |
|---|---|
| Omar Osama | Logic Coverage |
| Mina Magdy | Logic Coverage |
| Mohmed Hamada | Another Coverages & Tool administration |
| Ahmed Saleh | Another Coverages & Tool administration |