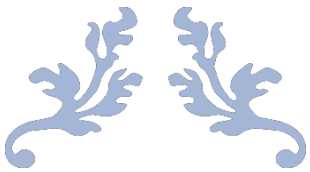

SOFTWARE ENGINEERING PROJECT-III

CUNIT Testing by Ronit Dahiya

3rd May 2019
Student Number :- 18204009

C-Unit Testing

What is C-Unit Testing?

It involves 3 stages :

- Creation of test first
- Start with simplest that works
- Incrementally add code while testing

The test serves as a benchmark and after that we optimise and refactorize without worry.

"Code that isn't tested doesn't work"

Result Test case for Maximum_Function.c

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


Suite: maximum_suite
  Test: MaxNum_Count_Empty_test ...
0.000000        FAILED
    1. main.c:36  - CU_ASSERT_EQUAL(max(array,0),0)
  Test: MaxNum_Count_Five_test ...
67.000000       passed
  Test: MaxNum_Count_One_test ...
5.000000        passed
  Test: MaxNum_Count_Negative_test ...
0.000000        FAILED
    1. main.c:73  - CU_ASSERT_EQUAL(max(array,4),-9)
  Test: MaxNum_Count_Eight_test ...
7.000000        passed
  Test: MaxNum_Count_Four_test ...
8.000000        passed

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests      6      6      4      2        0
             asserts      6      6      4      2      n/a

Elapsed time =    0.000 seconds

RUN SUCCESSFUL (total time: 164ms)
```

Test 1 - Fail

```
/**********************unit test cases************************/
/*
 * checking max function on an empty array
 * expected value : 0/garbage value
 * actual value : 0
 * test fail
 */
void MaxNum_Count_Empty(void) {
    double array[] = {};
    printf("\n%lf\t", max(array,0));
    CU_ASSERT_EQUAL(max(array,0),0); // assert statement to check max function
}
```

This test failed because the expected value was 0/garbage value but function returns always 0. As the max variable in *max* function is only declared it could be having any value.

Test 2 - Pass

```
/*
 * checking max function on an array having five elements
 * array : [ 4,5,67,12,7]
 * expected value : 67
 * actual value : 67
 * test pass
 */

void MaxNum_Count_Five(void) {
    double array[] = {4,5,67,12,7};
    printf("\n%lf\t", max(array,5));
    CU_ASSERT_EQUAL(max(array,5),67);  // assert statement to check max function
}
```

This test passed because the expected value was 67 and actual value also came as 67.
When *max* function is called with this array as an argument, max variable changed to 4 as it is always greater than 0 and after that it iterates through array and get the value 67 as max.

Test 3 - Pass

```
/*
 * checking max function on an array having one element
 * array : [5]
 * expected value : 5
 * actual value : 5
 * test pass
 */
void MaxNum_Count_One(void) {
    double array[] = {5};
    printf("\n%lf\t", max(array,1));
    CU_ASSERT_EQUAL(max(array,1),5); // assert statement to check max function
}
```

This test passed because the expected value was 5 and actual value also came as 5.
When *max* function is called with this array as an argument, max variable changed to 5 as it is always greater than 0 and thus return 5.

Test 4 - Fail

```
/*
 * checking max function on an array having four negative values
 * array : [-10,-22,-31,-9]
 * expected value : -9
 * actual value : 0
 * test fail
 */
void MaxNum_Count_Negative(void) {
    double array[] = {-10,-22,-31,-9};
    printf("\n%lf\t", max(array,4));
    CU_ASSERT_EQUAL(max(array,4),-9);  // assert statement to check max function
}
```

This test failed because the expected value was -9 but function returns 0 which is actual value.
As *max* function is called with this array, it max variable never got changed to negative value as its initialization value is 0, thus it return 0

Test 5 - Pass

```
/*
 * checking max function on an array having eight elements (both positive and negative)
 * array : [ 0,-5,6,7,2,-6,-10,2]
 * expected value : 7
 * actual value : 7
 * test pass
 */
void MaxNum_Count_Eight(void) {
    double array[] = {0,-5,6,7,2,-6,-10,2};
    printf("\n%lf\t", max(array,8));
    CU_ASSERT_EQUAL(max(array,8),7);  // assert statement to check max function
}
```

This test passed because the expected value was 7 and actual value also came as 7.
When *max* function is called with this array as an argument, max variable changed to 6 first as max variable only get changed to values greater than 0 due to the function logic and then it got changed to 7 which it returns.

Test 6 - Pass

```
/*
 * checking max function on an array having four elements
 * array : [ 3,5,8,1]
 * expected value : 8
 * actual value : 8
 * test pass
 */
void MaxNum_Count_Four(void) {
    double array[] = {3,5,8,1};
    printf("\n%lf\t", max(array,4));
    CU_ASSERT_EQUAL(max(array,4),8);  // assert statement to check max function
}
```

This test passed because the expected value was 8 and actual value also came as 8.
When *max* function is called with this array as an argument, max variable changed to 3,5 and next to 8, but after that it doesn't change and return 8.

Run Test

```
/************************unit test cases end************************/

// function to run the test cases
void runTests() {
    CU_initialize_registry(); // initializing CUnit registry
    CU_pSuite suite = CU_add_suite("maximum_suite",0,0); // Creating CUnit suite

    // Adding test cases to test suite (suite)
    CU_add_test(suite,"MaxNum_Count_Empty_test",MaxNum_Count_Empty);
    CU_add_test(suite,"MaxNum_Count_Five_test",MaxNum_Count_Five);
    CU_add_test(suite,"MaxNum_Count_One_test",MaxNum_Count_One);
    CU_add_test(suite,"MaxNum_Count_Negative_test",MaxNum_Count_Negative);
    CU_add_test(suite,"MaxNum_Count_Eight_test",MaxNum_Count_Eight);
    CU_add_test(suite,"MaxNum_Count_Four_test",MaxNum_Count_Four);

    // Setting CUnit modes (VERBOSE)
    CU_basic_set_mode(CU_BRM_VERBOSE);
    // Running the test suite
    CU_basic_run_tests();
    // Cleaning CUnit registry
    CU_cleanup_registry();
}

// Main function
int main() {
   // calling runTest function for test execution
   runTests();
   return 0;
}
```

Result Test case for Average_Function.c

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


Suite: average_suite
  Test: AvgNum_Count_Empty_test ...
-nan     FAILED
    1. main.c:37  - CU_ASSERT_EQUAL(isnan(average(array,0)),0)
  Test: AvgNum_Count_One_test ...
-nan     FAILED
    1. main.c:50  - CU_ASSERT_EQUAL(average(array,1),5)
  Test: AvgNum_Count_Two_test ...
3.000000         FAILED
    1. main.c:63  - CU_ASSERT_EQUAL(average(array,2),5.5)
  Test: AvgNum_Count_LastElementZero_test ...
4.400000         passed
  Test: AvgNum_Count_Four_test ...
0.375000         FAILED
    1. main.c:89  - CU_ASSERT_EQUAL(average(array,4),1.5)
  Test: AvgNum_Count_Negative_Positive_test ...
0.000000         FAILED
    1. main.c:102  - CU_ASSERT_EQUAL(average(array,4),-2.25)

Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      1      1    n/a      0        0
               tests      6      6      1      5        0
             asserts      6      6      1      5      n/a

Elapsed time =    0.000 seconds

RUN SUCCESSFUL (total time: 140ms)
```

Test 1 - FAIL

```
/****************************** unit test cases ***************************************/
/*
* checking average function on empty array
* array: []
* expected: NAN
* actual: NAN
* test passes
*/
void AvgNum_Count_Empty(void) {
    double array[]={};
    printf("\n%lf\t", average(array,0));
    CU_ASSERT_EQUAL(isnan(average(array,0)),0); // assert statement to check average functior
}
```

This test failed because the expected value was NaN (0/0 is not defined) and function returns Nan. The reason for fail test still unknown as output expected and actual are same.

Test 2 - FAIL

```
/*
* checking average function on single element array
* array: [5]
* expected: 5
* actual: 0
* test fails
*/
void AvgNum_Count_One(void) {
    double array[] = {5};
    printf("\n%lf\t", average(array,0));
    CU_ASSERT_EQUAL(average(array,1),5); // assert statement to check average function
}
```

This test failed because the expected value was 5 and actual value also came as 0.
When *avg* function is called with this array as an argument, according to function definition 'i' get iterated one less than the size so in this case it doesn't enter the for loop and hence fails.

Test 3 - FAIL

```
/*
* checking average function on two element array
* array: [6,5]
* expected: 5.5
* actual: 3
* test fails
*/
void AvgNum_Count_Two(void) {
    double array[] = {6,5};
    printf("\n%lf\t", average(array,2));
    CU_ASSERT_EQUAL(average(array,2),5.5); // assert statement to check average function
}
```

This test failed because the expected value was 5.5 and actual value came as 3.

When *avg* function is called with this array as an argument, according to function definition 'i' get iterated one less than the size, so in this case it doesn't consider 5 in the average calculation and hence fails.

Test 4 - PASS

```
/*
* checking average function on array having last element as 0
* array: [1,4,8,9,0]
* expected: 4.4
* actual: 4.4
* test passes
*/
void AvgNum_Count_LastElementZero(void) {
    double array[] = {1,4,8,9,0};
    printf("\n%lf\t", average(array,5));
    CU_ASSERT_EQUAL(average(array,5),4.4); // assert statement to check average function
}

/*
```

This test pass because the expected value was 4.4 and actual value came as 4.4.

When *avg* function is called with this array as an argument, according to function definition 'i' get iterated one less than the size, so in this case it doesn't consider 0 in the average calculation and hence the calculation is unaltered with its presence i.e makes not difference even if its included. Hence passes test

Test 5 - FAIL

```
/*
* checking average function on array having four elements
* array: [1.5, 0,0,4.5]
* expected: 1.5
* actual: 0.375
* test fails
*/
void AvgNum_Count_Four(void) {
    double array[] = {1.5,0,0,4.5};
    printf("\n%lf\t", average(array,4));
    CU_ASSERT_EQUAL(average(array,4),1.5); // assert statement to check average function
}
```

This test failed because the expected value was 1.5 and actual value came as 0.375.

When *avg* function is called with this array as an argument, according to function definition 'i' get iterated one less than the size, so in this case it doesn't consider 4.5 in the average calculation and hence fails.

Test 6 - FAIL

```
/*
* checking average function on an array having both positive and negative elements
* array: []
* expected: -2.25
* actual: 0
* test pass
*/
void AvgNum_Count_Negative_Postive(void) {
    double array[] = {2,3,-5,-9};
    printf("\n%lf\t", average(array,4));
    CU_ASSERT_EQUAL(average(array,4),-2.25); // assert statement to check average function
}
```

This test failed because the expected value was -2.25 and actual value came as 0.

When *avg* function is called with this array as an argument, according to function definition 'i' get iterated one less than the size, so in this case it doesn't consider -9 in the average calculation and hence fails.

Run test

```
/********************************* unit test cases end*****************************************/

// function to run all the test cases
void runTests() {
    CU_initialize_registry(); // intializing the CUnit registry
    CU_pSuite suite = CU_add_suite("average_suite",0,0); // creating CUnit test suite

    // adding test cases to the test suite (suite)
    CU_add_test(suite,"AvgNum_Count_Empty_test",AvgNum_Count_Empty);
    CU_add_test(suite,"AvgNum_Count_One_test",AvgNum_Count_One);
    CU_add_test(suite, "AvgNum_Count_Two_test", AvgNum_Count_Two);
    CU_add_test(suite, "AvgNum_Count_LastElementZero_test", AvgNum_Count_LastElementZero);
    CU_add_test(suite,"AvgNum_Count_Four_test",AvgNum_Count_Four);
    CU_add_test(suite,"AvgNum_Count_Negative_Positive_test",AvgNum_Count_Negative_Postive);

    // CUnit mode set
    CU_basic_set_mode(CU_BRM_VERBOSE);
    // running the test suite
    CU_basic_run_tests();
    // cleaning up the CUnit registry after running the test
    CU_cleanup_registry();
}

// main function
int main() {

  // calling runTests function to execute the tests on average function
  runTests();
  return 0;
}
```

****

END OF DOCUMENT