COMP10050: Software Engineering Project 1

Assignment 3: Unit Testing

Ashraf Ali - 19315281

Summary:

```
Run Summary:    Type  Total    Ran Passed Failed Inactive
              suites      2      2    n/a      0        0
               tests      8      8      1      7        0
             asserts     19     19      8     11      n/a

Elapsed time =    0.000 seconds
```

Average Function Tests

Test 1: Testing Basic Values for the Average function FAIL

```
/*Tests basic inputs for average*/
void average_test_normal() {
    double test1[] = {1.1, 2.2, 3.3, 4.4};
    CU_ASSERT_DOUBLE_EQUAL( actual: average(test1, 4),  expected: 2.75,  granularity: 0.0001);
    double test2[] = {5.6, 1.2, 4.3};
    CU_ASSERT_DOUBLE_EQUAL( actual: average(test2, 3),  expected: 3.7,  granularity: 0.0001);
    double test3[] = {1, 2, 3, 4, 5, 6};
    CU_ASSERT_DOUBLE_EQUAL( actual: average(test3, 6),  expected: 3.5,  granularity: 0.0001);
}
```

Reason:

The for loop for computing the sum only sums the first n-1 terms.

```
for(int i =0; i < size - 1; i++){
```

Test 2: Testing identical values for the Average Function FAIL

```
/*Tests when all values are the same*/
void average_test_same() {
    double test1[10];
    for (int i = 0; i < 10; i++) {
        test1[i] = 1.618;
    }
    CU_ASSERT_DOUBLE_EQUAL( actual: average(test1, 10), expected: 1.618, granularity: 0.0001);
    double test2[15];
    for (int i = 0; i < 15; i++) {
        test2[i] = 3.1415;
    }
    CU_ASSERT_DOUBLE_EQUAL( actual: average(test2, 15), expected: 3.1415, granularity: 0.0001);
}
```

Reason:

It divides the sum of n-1 elements by n, hence it always outputs a wrong value for the identical inputs.

```
return sum/size;
```

Test 3: Testing empty inputs for the Average Function PASSED

```
/*Tests when there are no values in the array*/
void average_test_empty() {
    double test1[] = {};
    CU_ASSERT( value: isnan( x: average(test1, 0)));
}
```

Reason:

Since the size of the input is 0, you can't divide by zero therefore we shouldn't expect a number.

```
return sum/size;
```

Test 4: Test single inputs for the Average function FAIL

```
void average_test_single() {
    double test1[] = {4};
    CU_ASSERT_DOUBLE_EQUAL( actual: average(test1, 1), expected: 4, granularity: 0.0001);
    double test2[] = {-0.404};
    CU_ASSERT_DOUBLE_EQUAL( actual: average(test2, 1), expected: -0.404, granularity: 0001)
}
```

Reason:

The for loop for computing the sum only sums the first n-1 terms. When it's only 1 input, it doesn't take the sum of the array.

```
for(int i =0; i < size - 1; i++){
```

Therefore we're diving 0 by 1 and can only get 0 for any input we have.

```
return sum/size;
```

Max Function Tests

Test 1: Testing Basic Values for the Max function FAIL

```
void max_test_normal() {
    double test1[] = {1.1, 2.2, 3.3, 4.4};
    CU_ASSERT_DOUBLE_EQUAL( actual: max(test1, 4),  expected: 4.4,  granularity: 0.0001);
    double test2[] = {5.6, 1.2, 4.3};
    CU_ASSERT_DOUBLE_EQUAL( actual: max(test2, 3),  expected: 5.6,  granularity: 0.0001);
    double test3[] = {1, 2, 3, 4, 5, 6};
    CU_ASSERT_DOUBLE_EQUAL( actual: max(test3, 6),  expected: 6,  granularity: 0.0001);
    double test4[] = {-0.1,-2,-3,-4,-5,-6};
    CU_ASSERT_DOUBLE_EQUAL( actual: max(test4, 6),  expected: -0.1,  granularity: 0.0001)
}
```

Reason:

The max variable is never defined, hence my compiler, mingw, sets it to zero.

```
double max;
```

Our 4th test uses negative values, the max variable never updates.

```
if(max < array[i])
    max = array[i];
```

Test 2: Testing Identical Values for the Max function FAIL

```
void max_test_same() {
    double test1[10];
    for (int i = 0; i < 10; i++) {
        test1[i] = 1.618;
    }
    CU_ASSERT_DOUBLE_EQUAL( actual: max(test1, 10),  expected: 1.618,  granularity: 0.0001);
    double test2[15];
    for (int i = 0; i < 15; i++) {
        test2[i] = -3.1415;
    }
    CU_ASSERT_DOUBLE_EQUAL( actual: max(test2, 15),  expected: 3.1415,  granularity: 0.0001);
    double test3[50];
    for (int i = 0; i < 50; i++) {
        test3[i] = 1.414213;
    }
    CU_ASSERT_DOUBLE_EQUAL( actual: max(test3, 50),  expected: 1.414213,  granularity: 0.0001);
}
```

Reason:

The max variable is only updated when a value in the array is greater than it.

```
if(max < array[i])
    max = array[i];
```

This works for arrays with values greater than or equal to 0, however our second test includes negative values.

Test 3: Testing Empty Values for the Max function FAIL

```
/*Tests when there are no values in the array*/
void max_test_empty() {
    double test1[] = {};
    CU_ASSERT_DOUBLE_EQUAL( actual: max(test1, 0),  expected: -INFINITY,  granularity: 0.0001);
}
```

Reason:

In theory the max value for an empty array is -INFINITY. The max variable is initialized to zero so it can't be -INFINITY.

```
double max;
```

Test 4: Testing Single Values for the Max function FAIL

```
void max_test_single() {
    double test1[] = {4};
    CU_ASSERT_DOUBLE_EQUAL( actual: max(test1, 1),  expected: 4,  granularity: 0.0001);
    double test2[] = {-0.404};
    CU_ASSERT_DOUBLE_EQUAL( actual: max(test2, 1),  expected: -0.404,  granularity: 0.0001);
    double test3[] = {3145.21};
    CU_ASSERT_DOUBLE_EQUAL( actual: max(test3, 1),  expected: 3145.21,  granularity: 0.0001);
}
```

Reason:

Again this would work if not for the fact that max is initialized to 0, hence any value less than 0 fails.

```
double max;
```