# Report.docx

Instructions: A file named report.doc or report.docx (in Microsoft Word format)  
or report.txt (an ordinary text file) that contains in addition your  
name and your UCLA Id Number :  
a. A brief description of notable obstacles you overcame.  
b. A list of the test data that could be used to thoroughly test your functions,  
along with the reason for each test. You must note which test cases your  
program does not handle correctly. (This could happen if you didn't have time  
to write a complete solution, or if you ran out of time while still debugging a  
supposedly complete solution.) Notice that most of this portion of your report  
can be written just after you read the requirements in this specification,  
before you even start designing your program.  
How nice! Your report this time doesn't have to contain any design documentation.  
As with Project 3, a nice way to test your functions is to use the assert facility from the  
standard library. As an example, here's a very incomplete set of tests for Project 4:

CARTER BALLOW: 806389359

Some of the difficulties that I encountered came out of the int countFloatingPointValues(const string array[], int n) function, and I found it hard because there were a lot of test cases to navigate and when I finally thought that I figured it out, it was just a few steps short. I finally cracked the code when I was able to nail the logic for identifying the first number and a decimal point and then if there are other characters then you can go ahead and return/ skipp to the next count and not consider it in the final results. Other than this, this project was super easy, and I just had to come up with a bunch of test cases with assert statements which was really tedious.

All of my test cases were in the main function that I will submit but there are a lot. These are all of them. ALL OF THESE ARE HANDLED CORRECTLY.

string data2[] = { "a", "b", "c" };//normal case with characters.

assert(locateMaximum(data2, 3) == 2); // "c" is the largest

string data1[] = { "12,", "98.76", "tyroin", "apple", "banana" };

assert(locateMaximum(data1, 5) == 2); // "tyroin" is lexicographically the largest because of the t, and the ints are not considered at all.

string datanew[] = { "12,", "98.76", "tyroin", "apple", "tyroin" };

assert(locateMaximum(data1, 5) == 2); // if there are two largest elements, return the first one

// Test with a single string

string dataSingle[] = { "a" };

assert(locateMaximum(dataSingle, 1) == 0); // Only one element, should return index 0

string dataIdentical[] = { "apple", "apple", "apple" };

assert(locateMaximum(dataIdentical, 3) == 0); // All identical strings, should return the first index

// Test countFloatingPointValues

string data3[] = { "3.14", "hello", "1.0", "foo", "5.67" };

assert(countFloatingPointValues(data3, 5) == 3); // 3 floating-point numbers: 3.14, 1.0, 5.67 should be considered which is three, just a normal case with strigns as well to make sure that they are not considered.

string dataNegative[] = { "-3.14", "hello", "-1.0", "foo", "5.67" };

assert(countFloatingPointValues(dataNegative, 5) == 3); // 3 floating-point numbers, including negative numbers

string data4[] = { "hello", "world", "no numbers" };

assert(countFloatingPointValues(data4, 3) == 0); // No floating-point values

string dataInvalid[] = { "12.34abc", "123.45", "12.34," };

assert(countFloatingPointValues(dataInvalid, 3) == 1); // Only "123.45" is valid

// Test hasNoCapitals

string data5[] = { "hello", "world", "no capitals" };

assert(hasNoCapitals(data5, 3) == true); // All lowercase, should return true

string data6[] = { "hello", "World", "no capitals" };

assert(hasNoCapitals(data6, 3) == false); // "World" has a capital letter, should return false

string dataMixed[] = { "hello", "world", "No Caps" };

assert(hasNoCapitals(dataMixed, 3) == false); // "No Caps" contains capitals, should return false

string dataSingleCapital[] = { "a", "B", "c" };

assert(hasNoCapitals(dataSingleCapital, 3) == false); // "B" is capital, should return false

string dataUpperCase[] = { "HELLO", "WORLD" };

assert(hasNoCapitals(dataUpperCase, 2) == false); // All caps, should return false

// Test shiftLeft

string data7[] = { "12,", "98.76", "tyrion", "a", "b" };

assert(shiftLeft(data7, 5, 2, "foo") == 2); // Shift by 2, placeholder "foo" should be used 2 times

assert(data7[3] == "foo" && data7[4] == "foo"); // Check last two elements

string data8[] = { "12,", "98.76", "tyrion", "a", "b" };

assert(shiftLeft(data8, 5, 10, "bar") == 5); // Shift by more than the length, placeholder "bar" should fill all positions

assert(data8[0] == "bar" && data8[1] == "bar" && data8[2] == "bar" && data8[3] == "bar" && data8[4] == "bar"); // Check all elements

assert(shiftLeft(data8, 5, -5, "foobar") == -1); // Invalid shift amount

assert(shiftLeft(data8, 0, 2, "foobar") == -1); // Edge case: empty array

// Test shifting by 0 (should not change the array)

string data9[] = { "one", "two", "three" };

assert(shiftLeft(data9, 3, 0, "foo") == 0); // No shift, no placeholders used

assert(data9[0] == "one" && data9[1] == "two" && data9[2] == "three"); // Array should remain unchanged

cout << "All tests passed!" << endl;