CS31 Project 4 Report

Benjamin He

UID: 804962948

**a. Notable Obstacles**

For the function countFloatingPointValues, my biggest difficulty was making sure that the function would be able to correctly decide whether each string in the array was a floating point value or not, specifically how to get it to recognize whether there is a number before or after a single decimal point. I finally decided to use a method like what I did for Project 3’s isValidEbayListingString function, where I used several Boolean variables to check whether certain conditions are true for the given string to be recognized as a float. In this project’s function, for every string that satisfied the all the conditions detailed in the project specifications to be a floating point value, there was a counter variable that tracked the number of floating point value strings, which was returned by the function.

The removeDuplicatedValues function also gave me a lot of trouble at first because I tried to eliminate the duplicated values and rearrange the elements of the array simultaneously, but I kept confusing myself, so instead I decided to separate deleting duplicate string and re-ordering the array into two parts. Deleting the duplicate string was really easy, but reordering the array proved to be quite troublesome because I wanted the function to be accurate even if an empty string was initially in the array, so that it wouldn’t be reordered at the end of the array if that empty string isn’t a duplicate. I solved this issue by changing the duplicated values into something other than an empty string, then reordering the array, then changing the duplicated values back into an empty string.

Finally, the shiftRight function was also a bit difficult to write because I wasn’t really sure how to shift all the elements in an array, so I had to plan out the logic of my program really meticulously to make sure that there wouldn’t be strange behaviors happening when the function is executed like looking past the array that it’s given and jumping into strange parts of the memory because my function had to look in elements of the input array before the current one in the for loop I used to reassign and “shift” the values of the array. Also, I initially planned to shift all the values and add in the placeholder strings, then count all the placeholder strings in the given array, but I realized that the number of placeholders used would be inaccurate if the original array had strings that were exactly the same as the placeholder string, so I just had the function count every time it shifted the array, and if amount was larger than the size of the given array, n, the number of placeholders used was the size of the input array.

**b. Test Data**

I wasn’t sure how I should format this test data so I just put in my code and put comments to explain each test case:

**//General test string arrays to test the functions**

string testString1[20] = { "jon-", "zaro'ld", "ja.cob", "k?aleb", "zachary", "jo.,sh", "l'arry", "ma\"rcus", "0.5", "3.2.0", "8d.24", "74", "89.3", "98.078w", "0", "92.", "32..0", "z", "z", "qwe" };

string testString2[7] = { "alex", "ben", "alex", "yuan", "hong", "alex", "yuan" };

string testString3[6] = { "alpha", "beta", "gamma", "gamma", "beta", "delta"};

string testString4[11] = { "0.5", "3.2.0", "8d.24", "074", "89.3", "98.078w", "0", "92.", "32..0", "z", "."};

string testString5[6] = { "", "apple", "", "potato", "apple", ""};

string testString6[7] = { "", "", "", "", "", "", "" };

string testString7[8] = { "b", "c", "a", "d", "A", "D", "C", "B"};

string testString8[8] = {"why?", "the-/", "I, ", "said:", "hey'", "yay;", "\"", "-" };

string testString9[5] = {"samwell", "jon", "margaery", "daenerys", "tyrion"};

string testString10[4] = {"howard-", "ucla.", "howard", "ucla"};

string testString11[4] = {"4.4.3.3", "44", "33.098", "33.098a"};

string testString12[4] = {"happy", "days", "happy", "days"};

**//Also had some string arrays to test specific functions:**

**//findMinimum-specific test string arrays**

string p[5] = { "l", "b", "k", "s", "t" };

string p1[4] = { "b", "c", "a", "d" };

string p2[4] = { "bEn", "mIchElLe", "KaThy", "JefFery" };

string p3[6] = { "a", "b", "c", "A", "B", "C" };

string p6[5] = { "l", "b", "k", "", "t" };

string p10[5] = { "bad", "cat", "ant", "dad", "ant" };

**//Testing the findMinimum function**

assert(findMinimum(testString1, 0) == -1); **//test invalid input for n recognition**

assert(findMinimum(testString9, 5) == 3); **//test accuracy based on spec example**

assert(findMinimum(testString1, 8) == 2); **//test accuracy with non-letter strings**

assert(findMinimum(testString7, 4) == 2); **//test accuracy with letter cases in**

**//partial array**

assert(findMinimum(testString7, 8) == 4); **//test accuracy with letter cases**

assert(findMinimum(p, 5) == 1); **//test accuracy with single letters**

assert(findMinimum(p6, 5) == 3); **//test accuracy with empty strings in array**

assert(findMinimum(p1, 4) == 2); **//test accuracy**

assert(findMinimum(p10, 5) == 2); **//test accuracy with words**

assert(findMinimum(p2, 4) == 3); **//test accuracy with strange cases**

assert(findMinimum(p3, -1) == -1); **//test invalid input recognition for n**

assert(findMinimum(p3, 0) == -1); **//test invalid input recognition for n**

**//countFloatingPointValues-specific test arrays**

string nums2[4] = { "1.0", "1", ".0101", "0.0123" };

string nums3[7] = { "40.5", "abs.5t", "4.5", "0.20.21", "10", "60.1a", "ar.er" };

string nums4[4] = { "4.", "4.5", "0.21", ".10" };

string nums5[4] = { ".a", "ab.", ".cd", "a.b" };

string nums6[4] = { "1.1", ".", "0.", ".9" };

string nums7[6] = { " 1.1", ".", "0.", " .9", "", "abc" };

**//Testing the countFloatingPointValues function**

assert(countFloatingPointValues(testString11, 4) == 2); **//test accuracy with**

**//multiple decimal points**

**//and letters**

assert(countFloatingPointValues(testString11, -14) == -1); **//test invalid n input**

**//recognition**

assert(countFloatingPointValues(testString1, 20) == 5); **//test accuracy with many**

**//non-floating point**

**//values**

assert(countFloatingPointValues(testString4, 11) == 5); **//test accuracy with array**

**//containing only single**

**//decimal point and**

**//different forms of**

**//floats**

assert(countFloatingPointValues(nums2, 4) == 4); **//test function accuracy**

assert(countFloatingPointValues(nums3, 7) == 3); **//test function accuracy**

assert(countFloatingPointValues(nums4, 4) == 4); **//test function accuracy**

assert(countFloatingPointValues(nums5, 4) == 0); **//test function accuracy**

assert(countFloatingPointValues(nums6, 4) == 3); **//test function accuracy**

assert(countFloatingPointValues(nums7, 5) == 1); **//test function accuracy**

assert(countFloatingPointValues(nums5, -1) == -1); **//test invalid array size**

**//recognition**

assert(countFloatingPointValues(nums5, 0) == -1); **//test invalid array size**

**//recognition**

**//countAllPunctuation-specific test string arrays**

string punc1[4] = { "h-ow-ard-", "uc'la.", "h.oward", "uc,la" };

string punc2[4] = { "billy!", ".h.am.", "13.20.@,", "!24/" };

string punc3[5] = { "12.34", "0.56", "100", "1.50", "67.5" };

string punc4[5] = { "1!2.3-4", "wh,at,", "10:0;", "bl?ah/'", "for\"" };

**//Testing the countAllPunctuation function**

assert(countAllPunctuation(testString10, -14) == -1); **//test invalid array size**

**//recognition**

assert(countAllPunctuation(punc4, -1) == -1); **//test invalid array size**

**//recognition**

assert(countAllPunctuation(punc4, 0) == -1); **//test invalid array size recognition**

assert(countAllPunctuation(testString10, 4) == 2); **//test function accuracy**

assert(countAllPunctuation(testString1, 8) == 8); **//test function accuracy**

assert(countAllPunctuation(testString8, 8) == 9); **//test function accuracy**

assert(countAllPunctuation(testString8, 8) == 9); **//test function accuracy**

assert(countAllPunctuation(punc1, 4) == 7); **//test function accuracy**

assert(countAllPunctuation(punc2, 4) == 9); **//test function accuracy**

assert(countAllPunctuation(punc3, 5) == 4); **//test function accuracy**

assert(countAllPunctuation(punc4, 5) == 11); **//test function accuracy**

**//replaceAll-specific test string arrays**

string replace[4] = { "happy", "days", "are here", "again" };

string r4[6] = { "does", "this", "function", "really", "work", "?" };

string r5[] = { "aaA", "aA", "BaaA", "Aaos" };

**//Testing the replaceAll function**

assert(replaceAll(testString9, 5, 'z', 'a') == 0); **//test function accuracy**

assert(replaceAll(testString9, -5, 'z', 'a') == -1); **//test invalid input array**

**//size**

**//recognition and return**

assert(replaceAll(testString9, 0, 'z', 'a') == -1); **//test invalid input array**

**//size**

**//recognition and return**

assert(replaceAll(replace, 4, 'a', 'z') == 5); **//test function accuracy**

assert(replaceAll(r4, 6, 'o', 'O') == 3); **//test function accuracy**

assert(replaceAll(r5, 4, 'a', 'y') == 6); **//test function accuracy**

assert(replaceAll(r5, 4, 'A', 'S') == 4); **//test function accuracy**

**//shiftRight-specific test string arrays**

string s1[] = { "pig", "dog", "dragon", "rooster", "rabbit", "mouse" };

string s2[] = { "pig", "dog", "dragon", "rooster", "rabbit", "mouse", "cat", "frog" };

**//Testing the shiftRight function**

assert(shiftRight(testString1, 7, 3, "foo") == 3); **//test function accuracy**

assert(shiftRight(testString1, 5, 25, "foo") == 5); **//test function accuracy when**

**//amount is greater than n**

assert(shiftRight(testString9, 5, 3, "foo") == 3); **//test function accuracy**

assert(shiftRight(testString9, -5, 3, "foo") == -1); **//test invalid n size**

**//recognition**

assert(shiftRight(testString9, 5, 5, "foo") == 5); **//test function accuracy when**

**//amount**

**//is equal to n**

assert(shiftRight(testString9, 5, 25, "foo") == 5); **//test function accuracy**

assert(shiftRight(s1, 6, 3, "animal") == 3); **//test function accuracy**

assert(shiftRight(s2, 8, 4, "boo") == 4); **//test function accuracy**

assert(shiftRight(s2, 8, 0, "boo") == 0); **//test function accuracy**

assert(shiftRight(s2, 8, 20, "boo") == 8); **//test function accuracy**

assert(shiftRight(s2, 0, 4, "boo") == -1); **//test invalid n size recognition**

assert(shiftRight(s2, 8, -20, "boo") == -1); **//test invalid amount input**

**//recognition**

**//removeDuplicatedValues-specific test string arrays**

string dup1[4] = { "happy", "happy", "days", "days" };

string dup2[8] = { "hi", "kathy", "whats", "kathy", "kathy", "up", "whats", "hi" };

string dup3[9] = { "will", "will", "you", "be", "you", "my", "valentine", "?", "my" };

string dup4[10] = { "does", "this", "does", "function", "does", "does", "really", "work", "function", "?" };

string dup5[11] = { "0.5", "3.2.0", "8d.24", "074", "89.3", "98.078w", "0", "92.", "z", "." };

string dup6[11] = { "0.5", "3.2.0", "0.5", "074", "89.3", "98.078w", "0.5", "92.", "z", "074" };

string dup7[9] = { "i", "want", "i", "want", "to", "eat", "to", "at", "bplate" };

string dup8[9] = { "i", "want", "i", "want", "to", "eat", "to", "at", "bplate" };

string dup9[3] = { "puppy", "", "apple" };

string dup10[5] = { "puppy", "", "puppy", "apple", "" };

string dup11[6] = { "puppy", "", "apple", "", "puppy", "" };

string dup12[6] = { "", "", "", "", "", "" };

string dup13[5] = { "a", "a", "", "a", "a" };

**//testing removeDuplicatedValues function**

assert(removeDuplicatedValues(testString3, 6) == 2); **//test function accuracy with**

**//spec example**

assert(removeDuplicatedValues(testString5, 6) == 3); **//test function accuracy when**

**//array contains empty**

**//strings**

assert(removeDuplicatedValues(testString6, 7) == 6); **//test array reordering with**

**//empty strings**

assert(removeDuplicatedValues(testString12, 4) == 2); **//test accuracy using spec**

**//example**

assert(removeDuplicatedValues(testString9, 5) == 0); **//test accuracy with spec**

**//example**

assert(removeDuplicatedValues(testString2, 7) == 3); **//test program accuracy**

assert(removeDuplicatedValues(dup3, 9) == 3); **//test function accuracy with**

**//various strings**

assert(removeDuplicatedValues(people, -5) == -1); **//test invalid n size**

**//recognition**

assert(removeDuplicatedValues(people, 0) == -1); **//test invalid n size recognition**

assert(removeDuplicatedValues(dup1, 4) == 2); **//test array reordering**

**//the rest of these just test function accuracy with various strings**

**//in the main, I also had a function to display the input string before and after running removeDuplicatedValues on it to make sure that it modified and reordered the array correctly**

assert(removeDuplicatedValues(dup2, 8) == 4);

assert(removeDuplicatedValues(dup4, 10) == 4);

assert(removeDuplicatedValues(dup6, 11) == 3);

assert(removeDuplicatedValues(dup7, 9) == 3);

assert(removeDuplicatedValues(dup8, 3) == 1);

assert(removeDuplicatedValues(dup9, 3) == 0);

assert(removeDuplicatedValues(dup10, 5) == 2);

assert(removeDuplicatedValues(dup11, 6) == 3);

assert(removeDuplicatedValues(dup12, 6) == 5);

assert(removeDuplicatedValues(dup13, 5) == 3);