Zachary Chang

UID: 805289445

Discussion 1G

Lecture 1

Programming Assignment 4 Report

1. One of the more notable obstacles that I had to overcome was getting my code to run in g++. My code would run fine with various test cases within XCode, but it would not run in g++ and would return an AddressSanitizer:DEADLYSIGNAL error. At first, I thought this was because I was comparing two string addresses instead of strings within my code, so I changed each comparison so that at least one of the strings was stored in a variable. This did not solve my solution, but I came across the answer while debugging my code. It turns out that one of my loops would try to access a value outside the range of the array. I fixed this off-by-one error, and my code was able to run in g++. Another obstacle I encountered was figuring out how to approach the “separator” function. After thinking about it, I realized that I could solve the problem by putting the whole array in alphabetical order which would satisfy any case.
2. (below) – Test case on the left, reason on the right

string names1[9] = {“carl”, “joe”, “bob”, “zeke”, “harold”, “ryan”, “terry”, “zeke”, “jaylen”};

*appendToAll*

|  |  |
| --- | --- |
| (names1, -5, “!!!”) | Check that the function returns -1 |
| (names1, 9, “!!!”) | Verify the function appends “!!!” to all strings |
| (names1, 4, “!!!”) | Verify the function only appends “!!!” to the first four strings |

*lookup*

|  |  |
| --- | --- |
| (names1, 9, “harold”) | Verify that the function finds the index of the string “harold” |
| (names1, 4, “harold”) | Verify that the function does not look outside its designated scope |
| (names1, 9, “zeke”) | Verify that the function returns the earliest instance of the string |
| (names1, -1, “bob”) | Check that the function returns -1 |

*positionOfMax*

|  |  |
| --- | --- |
| (names1, 9) | Verify that the function returns the earliest instance of the max string |
| (names1, -1) | Check that the function returns -1 |
| (names1, 0) | Check that the function returns -1 when no interesting values exist |
| (names1, 3) | Verify that the function searches within the first 3 values |

*rotateLeft*

|  |  |
| --- | --- |
| (names1, -4, 3) | Check that the function returns -1 |
| (names1, 3, -2) | Check that the function returns -1 |
| (names1, 9, 5) | Verify that the function successfully rotates the target string to the end |
| (names1, 6, 3) | Verify that the function only rotates within the subsequence |

string names2[8] = {“joe”, “bob”, “bob”, “carl”, “carl”, “carl”, “zak”, “harold”};

*countRuns*

|  |  |
| --- | --- |
| (names2, -1) | Check that the function returns -1 |
| (names2, 0) | Check that the function returns 0 |
| (names2, 6) | Verify that the function only counts within its designated scope |
| (names2, 8) | Verify that the function counts the number of runs in the whole array |

*flip*

|  |  |
| --- | --- |
| (names1, -6) | Check that the function returns -1 |
| (names1, 4) | Verify that the function only flips the first 4 elements |
| (names1, 9) | Verify that the function flips all the elements in the array |

string names3[9] = {“bob”, “larry”, “amy”, “jack”, “chris”, “ronald”, “willis”, “George”, “joe”};

string names4[6] = {“bob”, “larry”, “amy”, “jack”, “harold”, “marvin”};

*differ*

|  |  |
| --- | --- |
| (names3, -5, names4, 5) | Check that the function returns -1 |
| (names3, 4, names4, -2) | Check that the function returns -1 |
| (names3, 6, names4, 4) | Verify that the function simply returns the smaller of n1 and n2 |
| (names3, 9, names4, 6) | Verify that the function returns the first position where the two array values are not equal |

string names5[9] = {“bob”, “larry”, “amy”, “jack”, “chris”, “ronald”, “larry”, “amy”, “jack”};

string names6[3] = {“larry”, “amy”, “jack”};

*subsequence*

|  |  |
| --- | --- |
| (names5, -1, names6, 4) | Check that the function returns -1 |
| (names5, 9, names6, -3) | Check that the function returns -1 |
| (names5, 1, names6, 3) | Check that the function returns -1 (impossible to have subsequence in a shorter names5 subsection) |
| (names5, 5, names6, 3) | Verify that the function returns the index of the first instance of the subsequence |
| (names5, 9, names6, 3) | Verify that the function still returns the index of the first instance of the subsequence |

string names7[8] = {“bob”, “larry”, “amy”, “jack”, “chris”, “ronald”, “greg”, “darren”};

string names8[4] = {“greg”, “chris”, “darren”, “jack”};

*lookupAny*

|  |  |
| --- | --- |
| (names7, -4, names8, 3) | Check that the function returns -1 |
| (names7, 4, names8, -2) | Check that the function returns -1 |
| (names7, 3, names8, 4) | Verify that the function only searches within its designated scope |
| (names7, 4, names8, 4) | Verify that the function is able to find “jack” |
| (names7, 4, names8, 3) | Verify that the function cannot find “jack” |
| (names7, 8, names8, 4) | Verify that the function only returns the first matching instance |

*separate*

|  |  |
| --- | --- |
| (names7, -1, “Donald”) | Check that the function returns -1 |
| (names7, 8, “donald”) | Verify that the function returns the index of the first string after “donald” |
| (names7, 8, “Zoo”) | Verify that the function simply returns 8 |
| (names7, 8, “chris”) | Verify that the function returns the index of “chris” |