Douglas Frattini Edwards

Professor David Smallberg

COM SCI 31

31 October 2022

**Part A: Notable Obstacles**

* The most significant obstacle was certainly the split function. I believe the implementation of it would have been fairly simple if I had created other arrays to help separate the data, but I was committed to creating a version of the function that worked only with the input array. I tried several separate designs but quickly realized that the seemingly simplest implementation was to ignore the splitter string and instead sort the entire list alphabetically. Doing so ensures that things are split as intended. The only drawback was having to then manually search for the first element in the array that was not < splitter
* The other significant issue was with rotateLeft. The basic implementation was simple enough and worked fine, I just misinterpreted the instructions and had thought that we were supposed to flip n elements starting at pos, where we were actually supposed to flip elements starting at pos that were in the first n elements of the entire array. This misinterpretation led to a significant amount of time spent on calculating the length of a string array that culminated in the realization that I had simply misread the prompt
* The only other issue was with trying to understand and fully cover all the different variations of invalid input, including bad values of n, substrings larger than their parent strings, and much more. Hopefully I was able to figure out all the significant ways to break this program.

To conclude: the most difficult part of this assignment was adhering to the one sentence offering bonus points for not utilizing additional arrays in the rotateLeft, flip, and split functions.

**Part B: Test Data**

Lookup

| Data Used | | |
| --- | --- | --- |
| string g[4] = { "rishi", "margaret", "liz", "theresa" };  string e[4] = { "gordon", "tony", "", "john" };  string h[7] = { "rishi", "margaret", "gordon", "tony", "", "john", "liz" };  string j[5] = {}; // Empty array  string k[5] = { "", "", "", "dog", "" }; // Array with empty strings  string odd[5] = { "a", "b", "c", "d", "e" };  string l[1] = { "Dog" };  string d[5] = { "margaret", "margaret", "margaret", "tony", "tony" };  string m[10] = { "", "Dog", "Dog", "Cat", "Cat", "Cat", "Dog", "Cat", "Animal", "Reptile" };  string f[3] = { "liz", "gordon", "tony" };  string n[10] = { "", "Dog", "Dog", "Cat", "Cat", "Cat", "Dog", "Dave", "Animal", "Reptile" };  string o[3] = { "Dog", "Dog"};  string p[3] = { "Dave", "Sandra", "Jamie"};  string q[5] = { "a", "b", "c", "d", "e" };  string r[5] = { "five", "dog", "", "", "c" };  string s[5] = { "e", "d", "b", "c", "a" }; | | |
| Test | Reason | Passed? |
| assert(lookup(h, 7, "john") == 5); | Basic functionality | Yes |
| assert(lookup(h, 7, "gordon") == 2); | Basic functionality | Yes |
| assert(lookup(h, 2, "gordon") == -1); | Test for item not in specified range | Yes |
| assert(lookup(h, 7, "rishi") == 0); | Basic functionality | Yes |
| assert(lookup(h, 0, "rishi") == -1); | Test for n = 0 | Yes |
| assert(lookup(h, 3, "") == -1); | Test for empty string not in specified range | Yes |
| assert(lookup(h, 7, "") == 4); | Test for empty string | Yes |
| assert(lookup(j, 0, "david") == -1); | Test for n = 0 | Yes |
| assert(positionOfMax(h, 7) == 3); | Basic functionality | Yes |
| assert(positionOfMax(h, 0) == -1); | Test n = 0 | Yes |
| assert(positionOfMax(j, 0) == -1); | Test with empty array | Yes |
| assert(positionOfMax(k, 5) == 3); | Test with array with empty strings | Yes |
| assert(rotateLeft(odd, 5, 0) == 0 && odd[0] == "b" && odd[4] == "a" && odd[2] == "d"); | Basic rotation of whole array | Yes |
| assert(rotateLeft(odd, 3, 0) == 0 && odd[0] == "c" && odd[2] == "b" && odd[4] == "a"); | Rotation of part of array | Yes |
| assert(rotateLeft(g, 4, 1) == 1 && g[1] == "liz" && g[3] == "margaret" && g[0] == "rishi"); | Rotation with non-zero position | Yes |
| assert(rotateLeft(j, 0, 0) == -1); | Test n = 0 | Yes |
| assert(rotateLeft(odd, 5, 6) == -1); | Test with pos outside of range | Yes |
| assert(differ(h, 4, g, 4) == 1); | Basic functionality | Yes |
| assert(differ(h, 1, g, 1) == 1); | Test full match because of limited range | Yes |
| assert(differ(h, 0, g, 0) == 0); | Test n = 0 | Yes |
| assert(appendToAll(g, 4, "?") == 4 && g[0] == "rishi?" && g[3] == "margaret?"); | Basic functionality | Yes |
| assert(appendToAll(l, 1, "go") == 1 && l[0] == "Doggo"); | Test just one item in array | Yes |
| assert(appendToAll(g, -5, "Dog") == -1); | Test n < 0 error | Yes |
| assert(appendToAll(g, 2, "!") == 2 && g[0] == "rishi?!" && g[3] == "margaret?"); | Appending to limited range of items | Yes |
| assert(countRuns(d, 5) == 2); | Basic functionality | Yes |
| assert(countRuns(d, 3) == 1); | Functionality with limited range | Yes |
| assert(countRuns(d, 0) == 0); | Test n = 0 | Yes |
| assert(countRuns(m, 10) == 7); | Test with large array with repeated runs of the same string | Yes |
| assert(countRuns(k, 5) == 3); | Test with empty strings | Yes |
| assert(flip(f, 3) == 3 && f[0] == "tony" && f[2] == "liz"); | Basic functionality | Yes |
| assert(flip(k, 5) == 5 && k[1] == "dog"); | Test with empty strings | Yes |
| assert(flip(k, 0) == 0); | Test with empty array | Yes |
| assert((flip(m, 10) == 10) && m[1] == "Animal"); | Test with larger, more complex array | Yes |
| assert(m[4] == "Cat" && m[3] == "Dog" && m[9] == ""); | Test with larger, more complex array | Yes |
| assert(flip(g, -5) == -1); | Test n<0 error | Yes |
| assert(subsequence(n, 10, o, 2) == 1); | Basic functionality | Yes |
| assert(subsequence(n, 10, p, 3) == -1); | Subsequence not in main array | Yes |
| assert(subsequence(n, 10, p, 1) == 7); | Changing result by limiting length of included subsequence with n | Yes |
| assert(subsequence(n, 10, p, 0) == 0); | Searching for n = 0 substring | Yes |
| assert(subsequence(n, 0, p, 3) == -1); | Empty main array | Yes |
| assert(lookupAny(h, 7, f, 3) == 2); | Basic functionality | Yes |
| assert(lookupAny(q, 5, r, 5) == 2); | Basic functionality | Yes |
| assert(lookupAny(q, 0, r, 5) == -1); | Empty a1 array test | Yes |
| assert(lookupAny(q, 5, r, 4) == -1); | Test for no matches due to limiting n2 | Yes |
| assert(lookupAny(j, 0, r, 5) == -1); | Empty main array | Yes |
| assert(lookupAny(q, 5, j, 0) == -1); | Empty search array test | Yes |
| assert(split(s, 5, "c") == 2); | Basic functionality | Yes |
| assert(s[0] == "a" && s[1] == "b" && s[2] == "c" && s[3] == "d" && s[4] == "e"); | Checking to see list is sorted properly | Yes |
| assert(split(s, 1, "c") == 1); | Splitting array with only 1 item | Yes |
| assert(split(h, 7, "liz") == 3); | Splitting larger array | Yes |
| assert(split(k, 5, "David") == 4 && k[4] == "dog"); | Splitting array with empty items | Yes |
| assert(split(k, 0, "danny") == 0); | Splitting empty array | Yes |