Reema Kshetramade

SID: 205328402

Project 4 Report

1. One obstacle I overcame was in the separate function, making sure that if the separator was in the array, it would be placed in between the elements that were greater than it and those that were less than it. Originally, the function did not check if there was any element that equaled the separator and simply classified it as greater than the separator which would not ensure that the separator would be in the middle.

Also, I had difficulty returning the lowest position of the max string in the array if two or more were equal because my implementation of the function originally reassigned the position of the max string to the current index if the current string was greater than or equal to the current max. This returned the highest index of the max string if multiple strings were equal to the max string. I fixed this by reassigning the index of the max position only if the current string was greater than the max string, because if the current and max string were equal than the current max position would be the lowest position of the max in the array.

1. Test cases that could thoroughly test the program include:
   * Normal parameters are passed to each function
   * string a[3] = {“apple”, “orange”, “banana”};

string b[2] = {“apple”, “banana”};

string c[5] = {“cherry”, “peach”, “peach”, “blueberry”, “coconut”};

appendToAll(a, 3, “?!”);

appendToAll(b, 1, “@@”);

lookup(a, 3, “orange”);

lookup(b, 1, “apple”);

positionOfMax(a, 3);

positionOfMax(c, 4);

rotateLeft(a, 3, 1);

rotateLeft(c, 5, 2);

countRuns(a, 3);

countRuns(c, 4);

flip(a, 3);

flip(c, 5);

differ(a, 3, b, 2);

differ(a, 3, c, 3);

subsequence(a, 3, b, 1);

subsequence(b, 2, a, 1);

lookupAny(a, 3, b, 2);

lookupAny(c, 5, a, 1);

separate(a, 3, “cantaloupe”);

* An array with a negative number of elements is passed to all the functions
  + string c[3] = {“pear”, “watermelon”, “canteloupe”};

string d[2] = {“banana”, “kiwi”};

appendToAll(c, -2, “?!”);

lookup(c, -1, “orange”);

positionOfMax(c, -5);

rotateLeft(c, -2, 1);

countRuns(c, -4);

flip(c, -10);

differ(c, -1, b, 2);

subsequence(c, 3, b, -1);

lookupAny(c, -3, b, -2);

separate(c, -3, “cantaloupe”);

* + This would make sure that all the functions are able to handle this error correctly and return -1.
  + An array of 0 elements is passed to appendToAll
  + string b[2] = {“apple”, “banana”};

appendToAll(b, 0, “!!”);

* + This would make sure that appendToAll works properly when passed an empty array, since this is not an error and would simply append the string to 0 elements in the array.
  + A target element that is not in the array is passed to lookup
  + string e[2] = {“apple”, “cantaloupe”};

lookup(e, 2, “pear”);

* + This would make sure that the function still returns a value and works normally if the target value is not found in the array.
  + An array of 0 elements is passed to lookup
  + string f[2] = {“orange”, “banana”};

lookup(f, 0, “orange”);

* + This would make sure that the function still works even when passed an empty array because this would there are zero elements that could match target so -1 is returned.
  + An array of 0 elements is passed to positionOfMax
  + string g[2] = {“kiwi”, “mango”};

positionOfMax(g, 0);

* + This makes sure that the function correctly identifies that there are no interesting elements in the array and the function returns -1.
  + An array with multiple of the same max in it are passed to positionOfMax
  + string h[4] = {“apple”, “orange”, “banana”, “orange”);

positionOfMax(h, 4);

* + This makes sure that the function returns position of the earliest occurrence of the max string.
  + A position to start rotation larger than the number of elements in the array is passed to rotateLeft
  + rotateLeft(h, 4, 5);
  + This makes sure that the function identifies this as an error because it would result in accessing an element outside the array, so the function should return -1.
  + A position less than 0 is passed to rotateLeft
  + rotateLeft(h, 4, -1);
  + This makes sure that this is identified as an error and the function returns -1 because this would access an element not in the array.
  + An array of size 0 is passed to countRuns
  + countRuns(h, 0)
  + This makes sure that the function identifies that if there are no elements, there are no consecutive element runs, but this is not an error so the function should return 0.
  + An array of size 0 is passed to flip
  + flip(h, 0);
  + This makes sure that this is not identified as an error and the function just returns 0 because there are no elements to reverse in the array, so the array is just unchanged.
  + Arrays of different length are passed to differ
  + j[3] = {“peach”, “nectarine”, “grape”};

k[2] = {“peach”, “nectarine”};

differ(j, 3, k, 2);

* + This makes sure that the array stops checking elements in the arrays once elements run out in the smaller array so the function does not attempt to access an element outside the array.
  + A second, subsequence array of larger size than the first array is passed to subsequence
  + m[2] = {“strawberry”, “apricot”};

n[3] = {“strawberry”, “apricot”, “blackberry”};

subsequence(m, 2, n, 3);

* + This makes sure that the function returns -1 because if the subsequence array is larger than the actual array, then it could never be found in the actual array.
  + An array of size 0 is passed as the sub array to subsequence
  + subsequence(m , 2, n, 0);
  + This makes sure that the function returns 0, not -1, because an array of 0 elements is always found at the start of any array.
  + Two arrays with no matching elements are passed to lookupAny
  + p[3] = {“blueberry”, “lime”, “raspberry”}

q[2] = {“pumpkin”, “cherry”};

lookupAny(p, 3, q, 2);

* + This makes sure that the function still returns a value, -1, even if there are no matching elements between the two functions.
  + An array with the separator string in it is passed to separate
  + r[3] = {“pineapple”, “lemon”, “cherry”};

separate(r, 3, “lemon”);

* + This makes sure that the separator is correctly placed between the elements that are less than the separator and those greater than it, and not placed randomly among the other elements that are not less than the separator.