Ethan Duong

UID: 805124044

Notable Obstacles:

I had difficulty keeping track of my nested loops at times. I also pondered whether some of my functions could be written without a nested loop. I am pretty sure that the runtime of some of my searches could have been made linear, but I decided to just stick with what worked. Some of my code seemed redundant as well so shaving off the fat was also an issue.

Example Test Data: Explanations on next page.

string a[6] = { "123", "456", "789", "13423", "1e44", "2359" };

string people[5] = { "jon", "mamabbcc!", "samwell,", "daenerys.", "tyrion" };

string folks[8] = {

"samwell", "jon", "margaery", "daenerys",

"tyrion", "sansa", "llewmas", "noj"

};

string b[5] = { "5", "4", "", "2", "15" };

string single[1] = { "racecar" };

assert(isOnlyDigits(a, 6) == false);

assert(isOnlyDigits(a, 3) == true);

assert(isOnlyDigits(b, 5) == false);

assert(isOnlyDigits(a, -1) == false);

assert(isInIncreasingOrder(people, 3) == true);

assert(isInIncreasingOrder(people, 5) == false);

assert(isInIncreasingOrder(single, 1) == true);

assert(isInIncreasingOrder(single, 0) == false);

assert(locateMinimum(people, 5) == 3);

assert(locateMinimum(single, 1) == 0);

assert(locateMinimum(single, -2) == -1);

assert(firstNonRepeatedCharacter(people, 5, 0) == 'j');

assert(firstNonRepeatedCharacter(people, 5, 1) == '!');

assert(firstNonRepeatedCharacter(single, 1, 0) == 'e');

assert(firstNonRepeatedCharacter(single, 1, -1) == '\0');

assert(countPunctuation(people, 5) == 3);

assert(countPunctuation(people, 1) == 0);

assert(countPunctuation(single, 1) == 0);

assert(countPunctuation(single, -1) == -1);

assert(highestOccurredCharacter(people, 5, 0) == 'j');

assert(highestOccurredCharacter(people, 5, 2) == 'l');

assert(highestOccurredCharacter(single, 1, 0) == 'r');

assert(highestOccurredCharacter(single, -1, 0) == '\0');

assert(hasReverse(folks, 8) == true);

assert(hasReverse(people, 5) == false);

assert(hasReverse(single, 1) == true);

assert(hasReverse(single, -1) == false);

I organized the test cases based on which functions they test. The last test case in each set is meant to test when the function has improper inputs. The other test cases are meant to check if the function does its job accurately with some edge cases thrown in to assure robustness. Common edge cases include when the function has to handle empty strings and when the function is only meant to check a portion of the array. The program is able to handle these test cases effectively.