1. Obstacle:

The most enormous obstacle happens after I wrote the separate function. After completing the separate function, I wrote several test cases to test the function, but it always pumped some errors out. I carefully reviewed my code but believed that I haven’t made any errors, so it really confused me. However, after writing other functions and came back to this question, I found that I mistakenly input the wrong outcome number into the test case. In a word, my test case is wrong and my codes are fine. After fixing the error and pay more attention to the test cases correctness I write, there are no huge obstacles later.

1. Tests:

string stuff[7] = { "mahi", "bon", "cous", "", "tar", "mur" , "tar"};

assert(locate(stuff,3, "mahi") == 0); //normal test for locate

assert(locate(stuff, 2, "") == -1); // the case is out of bound

assert(locate(stuff, 4, "") == 3); // find the empty string in array

assert(locate(stuff, 0, "assd") == -1); //zero situation

assert(locate(stuff, 2, "hahah") == -1);//not found

assert(locate(stuff, 7, "tar") == 4); // more than one

assert(locate(stuff, -3, "tar") == -1);//invalid input

assert(locate(stuff, 7, "Bon") == -1); //different char

assert(reduplicate(stuff, 6) == 6); // normal test for reduplicate

assert(stuff[2] == "couscous"); //check if the reduplicate process is successful

assert(stuff[3] == ""); //check if the the reduplicate process if sucessful

string cast[6] = { "elsa", "ariel", "mulan", "tiana", "belle", "moana" };

assert(locationOfMax(cast, 6) == 3); // normal test

string cast3[5] = {"elsa", "ariel", "mulan", "tina", "tina"};

assert(locationOfMax(cast3, 2) == 0); // normal test

assert(locationOfMax(cast3, 5) == 3); //more than one

assert(locationOfMax(cast3, -2) == -1); //invalid input

assert(locationOfMax(cast3, 0) == -1); //no elements

string folks[5] = { "moana", "elsa", "ariel", "raya", "mulan" };

assert(circleLeft(folks, 5, 1) == 1); //check if the return value is correct

string folks2[6] = {"lily", "hello", "sam", "cat", "ray", "new"};

assert(circleLeft(folks2, 3, 1) == 1);

assert(folks2[4] == "ray"); //check if the circleLeft functions work correctly

assert(folks2[1] == "sam"); //check if sam is in the end now

assert(folks2[2] == "hello"); // check if other string is moving forward

assert(circleLeft(folks2, 0, 0) == -1 && folks2[2] == "hello" );//zero test

assert(circleLeft(folks2, -9, 1) == -1);//negative input

assert(circleLeft(folks, 2, 7) == -1);//greater test

string d[9] = {"belle", "merida", "raya", "raya", "tiana", "tiana", "tiana", "raya", "raya"};

assert(enumerateRuns(d, 9) == 5); // normal test

string e[5] = {"abc", "dee", "dee"};

assert(enumerateRuns(e, 3) == 2); //normal test

assert(enumerateRuns(e, 1) == 1); //normal test

string f[0] = {};

assert(enumerateRuns(f, 0) == 0); // empty array

assert(enumerateRuns(d, -6) == -1);//invalid input

assert(enumerateRuns(d, 0) == 0);// input is zero

string roles[6] = { "merida", "raya", "", "belle", "moana", "elsa" };

assert(flip(roles, 4) == 4);

assert(roles[1] == ""); //normal test

assert(roles[5] == "elsa"); //test the object that is not been changed

assert(roles[0] == "belle"); // test the normal test effect

assert(flip(roles, 0) == 0 && roles[0] == "belle"); //zero test

assert(flip(roles,-8) == -1 && roles[1] == ""); //invalid input

string roles2[6] = { "merida", "raya", "", "belle", "moana", "elsa" };

string group[5] = { "merida", "raya", "elsa", "", "belle" };

assert(locateDifference(roles2, 6, group, 5) == 2); //normal situation

assert(locateDifference(roles2, 2, group, 1) == 1); //normal situation

assert(locateDifference(roles2, 2, group, 2) == 2);//equal situation

assert(locateDifference(roles2, 1, group, 2) == 1); //return the smaller situation

assert(locateDifference(roles2, 0, group, 3) == 0); //zero test

assert(locateDifference(roles2, -3, group, 2) == -1);//invalid input

string names[10] = { "moana", "mulan", "raya", "tiana", "merida" };

string names1[10] = { "mulan", "raya", "tiana" };

string cat[6] = {"abc", "bcd", "sdf", "tiana", "sdf", "tiana"};

string dog[2] = {"sdf", "tiana"};

assert(subsequence(names, 5, names1, 3) == 1);

string names2[10] = { "moana", "tiana" };

assert(subsequence(names, 4, names2, 2) == -1);

string empty[0] = {};

string empty2[0] = {};

assert(subsequence(empty, 0, empty, 0) == 0); // all arrays are empty arrays

assert(subsequence(cat, 4, dog, 0) == 0);//zero test

assert(subsequence(names, 4, dog, 4) == -1);//no found

assert(subsequence(cat, 4, dog, 2) == 2); //normal test with multiple equal terms

assert(subsequence(cat, 1, dog, 1) == -1);//no found

string names3[10] = { "moana", "mulan", "raya", "tiana", "merida" };

string set1[10] = { "elsa", "merida", "tiana", "mulan" };

assert(locateAny(names3, 6, set1, 4) == 1);

string set2[10] = { "belle", "ariel" };

assert(locateAny(names3, 6, set2, 2) == -1);

assert(locateAny(names3, -4, set2, 3) == -1);//invalid input

assert(locateAny(names3, 2, set2, 0) == -1); //zero test

assert(locateAny(names3, 0, set2, 0) == -1); //both zero test

string cast1[7] = { "elsa", "ariel", "mulan", "belle", "tiana", "moana" , ""};

assert(separate(cast1, 6, "merida") == 3); //normal test

string cast2[4] = { "mulan", "tiana", "ariel", "raya" };

assert(separate(cast2, 4, "raya") == 2); // normal test

assert(separate(cast2, 3, "") == 0);//all not smaller test(first one test)

assert(separate(cast1, 5, "ariel") == 0);//the word itself is the smallest;

assert(separate(cast2, 4, "tiana") == 3); // the word itself is the largest

string same[4] = {"a", "a", "a", "a"};

assert(separate(same, 4, "a") == 0); // all the same test

assert(separate(cast2, -3, "ca") == -1); //invalid input

assert(separate(cast2, 0, "case") == 0); // zero input

cerr << "ALL TEST PASSED";