Amy Yu

905138432

Project 4 Report

1. One problem that I had was finding a way to write the program without creating a new array. I was able to solve this by creating temporary variables to hold the string and then swapping them. Another issue I had was trying to figure out the conditions for the loops and knowing when to break out of the loop. I also had some difficulty deciding what to do if the size of the array was given to be zero, because this required different outputs for different functions.
2. Test Cases

string h[7] = { "romanoff", "thor", "rogers", "banner", "", "danvers", "rogers" };

assert(countMatches(h, 7, "rogers") == 2); *// checks for correct number of matches*

assert(countMatches(h, 0, "") == 0); *// return 0 when n = 0*

assert(countMatches(h, 5, "") == 1); *// checks if empty string is counted*

assert(countMatches(h, -1, "thor") == -1); *// checks if negative input for n returns -1*

assert(countMatches(h, 7, "rhodes") == 0); *// checks for no matches*

assert(countMatches(h, 0, "rogers") == 0); *// checks that no strings are looked at*

assert(detectMatch(h, 7, "rogers") == 2); *// checks for correct position*

assert(detectMatch(h, 2, "rogers") == -1); *// checks if only the first two strings in the array are looked at*

**int** bg;

**int** en;

assert(detectSequence(h, 7, "banner", bg, en) && bg == 3 && en == 3); *// checks for correct output*

assert(!detectSequence(h, -7, "banner", bg, en)); *// checks if negative input for n returns false*

assert(!detectSequence(h, 0, "banner", bg, en)); *// checks if negative input for n returns false*

assert(!detectSequence(h, 7, "stark", bg, en)); *// checks if it returns false if the string is not in the array*

string b[7] = { "romanoff", "rogers", "rogers", "rogers", "", "danvers", "rogers" };

assert(detectSequence(b, 7, "rogers", bg, en) && bg == 1 && en == 3); *// checks for correct output when there are consecutive duplicate strings*

string g[4] = { "romanoff", "thor", "banner", "danvers" };

assert(detectMin(g, 4) == 2); *// checks for correct position of min*

assert(detectMin(b, 7) == 4); *// checks empty string as min*

assert(detectMin(b, 0) == -1); *// checks if returns -1 if no strings are examined*

assert(detectDifference(h, 4, g, 4) == 2); *// g and h are different at position 2*

assert(detectDifference(h, 0, g, 4) == -1); *// returns -1 if the size is 0*

assert(contains(h, 7, g, 4)); *// h contains all elements in g in the same order*

assert(!contains(h, 7, b, 7)); *// h does not contain all elements in b in the same order*

assert(!contains(h, -1, b, 7)); *// cannot have negative size*

assert(!contains(h, 0, b, 7)); *// a1 cannot have zero elements*

assert(contains(h, 7, h, 7)); *// true if comparing the same array*

assert(moveToBack(g, -4 , 1) == -1 && g[1] == "thor" && g[3] == "danvers"); *// array is unchanged if size is negative*

assert(moveToBack(g, 0 , 1) == -1 && g[1] == "thor" && g[3] == "danvers"); *// returns -1 if n = 0*

assert(moveToBack(g, 4 , -1) == -1 && g[1] == "thor" && g[3] == "danvers"); *// array is unchanged if position is negative*

assert(moveToBack(g, 4, 1) == 1 && g[1] == "banner" && g[3] == "thor"); *// checks for correct output*

assert(moveToBack(b, 7, 0) == 0 && b[0] == "rogers" && b[6] == "romanoff"); *// if position equal zero*

string f[4] = { "danvers", "banner", "thor", "rogers" };

assert(moveToFront(f, -4, 2) == -1 && f[0] == "danvers" && f[2] == "thor"); *// array is unchanged if size is negative*

assert(moveToFront(f, 4, -2) == -1 && f[0] == "danvers" && f[3] == "rogers"); *// array is unchanged if position is negative*

assert(moveToFront(f, 0, 2) == -1 && f[0] == "danvers" && f[3] == "rogers"); *// returns -1 if n = 0*

assert(moveToFront(f, 4, 2) == 2 && f[0] == "thor" && f[2] == "banner"); *// checks for correct output*

string e[5] = { "danvers", "danvers", "danvers", "thor", "thor" };

assert(deleteDups(e, 0) == 0 && e[1] == "danvers"); *// returns 0 if n = 0 and leaves array unchanged*

assert(deleteDups(e, 5) == 2 && e[1] == "thor"); *// checks if duplicates are deleted*

string c[7] = { "romanoff", "rogers", "rogers", "", "", "danvers", "rogers" };

assert(deleteDups(c, 7) == 5 && c[1] == "rogers"); *// checks if empty string is deleted*

string x[4] = { "rhodes", "rhodes", "tchalla", "thor" };

string y[4] = { "banner", "danvers", "rhodes", "rogers" };

string z[10];

string z1[10];

string z2[4];

assert(meld(x, 4, y, 4, z, 10) == 8 && z[5] == "rogers");

assert(meld(x, 2, y, 2, z1, 10) == 4 && z[0] == "banner");

assert(meld(x, 4, y, 4, z2, 4) == -1); *// if result has more than max elements*

assert(meld(x, 0, y, 0, z2, 0) == 0); *// if two empty arrays are added*

assert(split(h, 0, "rogers") == 0); *// returns 0 if empty array*

assert(split(h, 7, "rogers") == 3);

string a[7] = { "romanoff", "rogers", "thor", "banner", "rogers", "rogers", "stark" };

assert(split(a, 4, "rogers") == 1);

string a1[6] = { "rhodes", "rhodes", "tchalla", "thor", "peter", "stark" };

assert(split(a1, 6, "banner") == 6);

assert(split(a1, 6, "tchalla") == 4);