Report

Li-Wei Tseng

304289977

1. Obstacles overcome:

When I tested the program, I followed the spec and thus use assert. I had to be careful to put the function which is passed by reference to the last. Otherwise, the previous program will be affected by it.

In addition, I have to be careful about the scope of loops and not put other things in the curly bracket if I do not want them to be affected by loops.

1. **Note: I did not create any additional array for functions** moveToend, firstInequality, removeCopies, **and** takeSides**.**
2. Test data

string h[7] = {"Phil", "Keegan", "Morgan", "Charl", "", "Stevie", "Morgan"};

assert(count(h, 5, "Morgan") == 1);

assert(count(h, 7, "") == 1);

assert(count(h, 7, "Jhonattan") == 0);

assert(count(h, 7, "phil") == 0);

assert(count(h, 0, "Morgan") == 0);

assert(count(h, -5, "Morgan") == -1);

assert(findFirst(h, 7, "Morgan") == 2);

assert(findFirst(h, 2, "Morgan") == -1);

assert(findFirst(h, 7, "morgan") == -1);

assert(findFirst(h, -3, "Morgan") == -1);

string team[5] = { "Phil", "Vijay", "Keegan", "Rickie", "Bubba" };

assert(moveToEnd(team, 5, 1) == 1 && team[0] == "Phil" && team[1] == "Keegan" && team[2] == "Rickie" && team[3] == "Bubba" && team[4] == "Vijay");

string g[4] = { "Phil", "Keegan", "Charl", "Stevie" };

assert(firstInequality(h, 4, g, 4) == 2);

assert(firstInequality(h, 0, g, 4) == 0);

assert(firstInequality(h, 4, g, 1) == 1);

assert(firstInequality(h, -4, g, -1) == -1);

string team2[5] = { "Phil", "Dustin", "Jhonattan", "Charl", "Rickie" };

string people[6] = { "Phil", "Dustin", "Vijay", "Bubba" };

assert(firstInequality(team2, 5, people, 4) == 2); // returns 2

assert(firstInequality(team2, 2, people, 1) == 1); // returns 1

string z[5] = { "Phil", "Keegan", "Charl","", "Stevie" };

string big[10] = { "Phil", "Charl", "Keegan", "Vijay", "Jhonattan", "Keegan" };

string little1[10] = { "Charl", "Vijay", "Jhonattan" };

bool b1 = subsequence(big, 6, little1, 3);

assert(b1 == true); // returns true

string little2[10] = { "Keegan", "Charl" };

bool b2 = subsequence(big, 6, little2, 2);

assert(b2 == false); // returns false

string little3[10] = { "Charl", "Keegan", "Keegan" };

bool b3 = subsequence(big, 6, little3, 3);

assert(b3 == true);

string little4[10] = { "Charl", "Charl", "Keegan" };

bool b4 = subsequence(big, 6, little4, 3);

assert(b4 == false);

bool b5 = subsequence(big, -6, little2, 2);

assert(b5 == false);

assert(moveToEnd(g, 4, 1) == 1 && g[1] == "Charl" && g[3] == "Keegan");

assert(moveToEnd(h, 5, 1) == 1 && h[1] == "Morgan" && h[4] == "Keegan");

assert(moveToEnd(g, -4, 1) == -1);

string e[5] = { "Charl", "Charl", "Charl", "Morgan", "Morgan" };

assert(removeCopies(e, 5) == 2 && e[1] == "Morgan" && e[0] == "Charl");

string d[9] = {"Keegan", "Charl", "Vijay", "Vijay", "Bubba", "Bubba", "Bubba", "Vijay", "Vijay"};

assert(removeCopies(d, 9) == 5 && d[0] == "Keegan" && d[1] == "Charl" && d[2] == "Vijay" && d[3] == "Bubba" && d[4] == "Vijay");

string f[6] = { "Bubba", "Morgan", "Quentin", "Charl", "Stevie", "Keegan" };

assert(takeSides(f, 6, "Phil") == 4);

string y[4] = { "Phil", "Morgan", "Stevie", "Keegan" };

assert(takeSides(y, 4, "Morgan") == 1);