Report

1. One notable obstacle was making sure my for loops were not indexing out of bounds. Since index out of bounds is illegal I was accessing values outside of the array length which caused errors. Another obstacle was finding the most efficient algorithm to implement because more complex algorithms where much harder to implement.
2. In addition to the test cases provided for int main() in the spec I used…

string d[9] = {

"thor", "romanoff", "parker", "parker", "stark", "stark", "stark", "parker", "parker"

};

assert(tally(d, 9, "parker") == 4 && tally(d, 5, "stark") == 1 && tally(d, 9, "tchalla") == 0);

to test int tally.

string people[5] = { "parker", "thor", "stark", "banner", "romanoff" };

assert(findMatch(people, 5, "banner") == 3);

to assert findMatch was working properly.

string people[5] = { "parker", "thor", "stark", "banner", "romanoff" };

assert(moveToEnd(people, 5, 1) == 1 && people[2] == "banner");

to assert moveToEnd was working properly.

string people[5] = { "parker", "thor", "stark", "banner", "romanoff" };

int j = moveToBeginning(people, 5, 2);

assert(j == 2 && people[1] == "parker");

to assert moveToBeginning was working properly.

string cast[5] = { "parker", "thor", "stark", "banner", "romanoff" };

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

int k = findDifference(cast, 5, roles, 4); // returns 2

int m = findDifference(cast, 2, roles, 1); // returns 1

assert(k == 2 && m == 1);

to assert findDifference was working properly.

string d[9] = {

"thor", "romanoff", "parker", "parker", "stark", "stark", "stark", "parker", "parker"

};

int p = eliminateDups(d, 9); // returns 5

// d[0] through d[4] now contain "thor" "romanoff" "parker" "stark" "parker"

// We no longer care what strings are in d[5] and beyond.

assert(p == 5 && d[1] == "romanoff");

to assert elimateDups was working properly.

string big[10] = { "parker", "thor", "stark", "banner", "romanoff", "stark" };

string little1[10] = { "thor", "banner", "romanoff" };

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

string little2[10] = { "stark", "thor" };

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

string little3[10] = { "thor", "stark", "stark" };

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

string little4[10] = { "thor", "thor", "stark" };

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

assert(b1 == true && b2 == false && b3 == true && b4 == false);

to assert subsequence was working properly.

string x[5] = { "banner", "rhodes", "rogers", "stark", "tchalla" };

string y[4] = { "parker", "rogers", "rogers", "thor" };

string z[20];

int n = makeMerger(x, 5, y, 4, z, 20); // returns 9

// z has banner parker rhodes rogers rogers rogers stark tchalla thor

assert(n == 9 && z[2] == "rhodes");

to assert makeMerger was working properly.

string f[6] = { "rhodes", "banner", "stark", "parker", "thor", "rogers" };

int r = separate(f, 6, "romanoff"); // returns 4

// f might now be

// "rhodes" "banner" "rogers" "parker" "thor" "stark"

// or "rogers" "parker" "banner" "rhodes" "stark" "thor"

// or several other orderings.

// The first 4 elements are < "romanoff"; the last 2 aren't.

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

int s = separate(g, 4, "rogers"); // returns 1

// g must now be either

// "banner" "rogers" "romanoff" "thor"

// or "banner" "rogers" "thor" "romanoff"

// All elements < "rogers" (i.e., "banner") come before all others.

// All elements > "rogers" (i.e., "thor" and "romanoff") come

// after all others.

assert(r == 4 && s == 1);

to assert separate was working properly.

string d[9] = {

"thor", "romanoff", "parker", "parker", "stark", "stark", "stark", "parker", "parker"

};

int b;

int e;

bool b1 = findRun(d, 9, "parker", b, e);

assert(b1 == true && b == 2 && e == 3);

to assert findRun was working properly.