1. No major obstacles during this project. The specifications were tricky for some functions.
2. Test Data:

*/\* ====== Testing countMatches ===== \*/*

cerr << "====== Testing countMatches =====" << endl;

string d[9] = {

"thor", "romanoff", "danvers", "danvers", "stark", "stark", "stark", "danvers", "danvers"

};

cerr << countMatches(d, 9, "danvers") << endl; *// returns 4*

cerr << countMatches(d, 5, "stark") << endl; *// returns 1*

cerr << countMatches(d, 9, "barton") << endl; *// returns 0*

*/\* ====== Testing detectMatches ===== \*/*

cerr << "====== Testing detectMatches =====" << endl;

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

cerr << detectMatch(people, 5, "banner") << endl; *// returns 3*

*/\* ====== detectSequence ===== \*/*

cerr << "====== detectSequence =====" << endl;

string t[9] = {

"thor", "romanoff", "danvers", "danvers", "stark", "stark", "stark", "danvers", "danvers"

};

**int** b;

**int** e;

*// returns true and sets b to 2 and e to 3*

cerr << detectSequence(t, 9, "danvers", b, e) << endl;

cerr << b << " " << e << endl;

*// returns true and sets b to 1 and e to 1*

cerr << detectSequence(t, 9, "romanoff", b, e) << endl;

cerr << b << " " << e << endl;

*// returns false and leaves b and e unchanged*

cerr << detectSequence(t, 9, "rogers", b, e) << endl;

cerr << b << " " << e << endl;

*/\* ====== detectMin ===== \*/*

cerr << "====== detectMin =====" << endl;

string peoples[5] = { "danvers", "thor", "stark", "banner", "romanoff" };

*// returns 3, since banner is earliest in alphabetic order*

cerr << detectMin(peoples, 5) << endl;

string Xen[7] = { "danvers", "apple", "thor", "stark", "banner", "apple", "romanoff" };

*//returns 1, since apple is earliest in alphabetic order AND the apple at index 1 is earlier than the apple at index 5*

cerr << detectMin(Xen, 7) << endl;

*/\* ====== moveToBack ===== \*/*

cerr << "====== moveToBack =====" << endl;

string peoplez[5] = { "danvers", "thor", "stark", "banner", "romanoff" };

cerr << moveToBack(peoplez, 5, 1) << endl; *// returns 1*

*// people now contains: "danvers" "stark" "banner", "romanoff" "thor"*

**for** (**int** k = 0; k < 5; k++)

cerr << peoplez[k] << ", ";

*/\* ====== moveToFront ===== \*/*

cerr << "====== moveToFront =====" << endl;

string peopley[5] = { "danvers", "thor", "stark", "banner", "romanoff" };

cerr << moveToFront(peopley, 5, 2) << endl;

*// people now contains: "stark" "danvers" "thor" "banner", "romanoff"*

**for** (**int** k = 0; k < 5; k++)

cerr << peopley[k] << ", ";

*/\* ====== dectectDifference ===== \*/*

cerr << "====== dectectDifference =====" << endl;

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

string roles[4] = { "danvers", "thor", "barton", "rhodes" };

cerr << detectDifference(cast, 5, roles, 4) << endl; *// returns 2*

cerr << detectDifference(cast, 2, roles, 1) << endl; *// returns 1*

*/\* ====== deleteDups ===== \*/*

cerr << "====== DeleteDups =====" << endl;

string dd[9] = {

"thor", "romanoff", "danvers", "danvers", "stark", "stark", "stark", "danvers", "danvers"

};

cerr << deleteDups(dd, 9) << endl; *// returns 5*

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

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

**for** (**int** j = 0; j != 9; j++)

cerr << dd[j] << ", ";

*/\* ====== contains ===== \*/*

cerr << "====== contains =====" << endl;

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

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

cerr << contains(big, 6, little1, 3) << endl; *// returns true*

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

cerr << contains(big, 6, little2, 2) << endl; *// returns false*

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

cerr << contains(big, 6, little3, 3) << endl; *// returns true*

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

cerr << contains(big, 6, little4, 3) << endl; *// returns false*

cerr << contains(big, 6, little4, 0) << endl; *// returns true*

*/\* ====== meld ===== \*/*

cerr << "====== meld =====" << endl;

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

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

string z[20];

cerr << meld(x, 5, y, 4, z, 20) << endl;*// returns 9*

*// z has banner danvers rhodes rogers rogers rogers stark tchalla thor*

**for** (**int** k = 0; k != 9; k++)

{

cerr << z[k] << ", ";

}

cerr << endl;

*/\* ====== split ===== \*/*

cerr << "====== split =====" << endl;

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

cerr << split(f, 6, "romanoff") << endl; *// returns 4*

*// f might now be*

*// "rhodes" "banner" "rogers" "danvers" "thor" "stark"*

*// or "rogers" "danvers" "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" };

cerr << split(g, 4, "rogers") << endl; *// 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.*

Additional Tests:

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

assert(countMatches(h, 7, "rogers") == 2);

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

assert(countMatches(h, 7, "rhodes") == 0);

assert(countMatches(h, 0, "rogers") == 0);

assert(detectMatch(h, 7, "rogers") == 2);

assert(detectMatch(h, 2, "rogers") == -1);

int bg;

int en;

assert(detectSequence(h, 7, "banner", bg, en) && bg == 3 && en == 3);

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

assert(detectMin(g, 4) == 2);

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

assert(contains(h, 7, g, 4));

assert(moveToBack(g, 4, 1) == 1 && g[1] == "banner" && g[3] == "thor");

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

assert(moveToFront(f, 4, 2) == 2 && f[0] == "thor" && f[2] == "banner");

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

assert(deleteDups(e, 5) == 2 && e[1] == "thor");

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

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

string z[10];

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

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

cout << "All tests succeeded" << endl;