Initially, one of the challenges I faced stemmed from the fact that I didn’t thoroughly read the spec, and thus was attempting to find the size of the array incase n was greater than it, which I didn’t know how to do. Upon rereading the spec, I realised that this was unnecessary for the purposes of this project. Soon after, I met a challenge at the deleteDups() function, which I didn’t know how to write initially, until I realised that I could do so with another array, although I could have gotten a bonus point had I not used one. I also thought contains was difficult initially and overthought it, thinking I would have to track the index of each string found as I progressed through the array to make sure that the values were in consecutive order (by comparing the indices to make sure the next one was always greater than the previous), which I soon realised was unnecessary. Finally, the split function caused me a bit of trouble, as I initially attempted to try swapping in order to not use additional arrays when organizing the given array a[] based around the splitter string, before finally succumbing to using additional arrays to effortlessly organize my original array. I realised that it would be useful to use the moveToFront() and moveToBack() methods, but I kept having issues with strings being moved around before I could get to them with my for loop, leading to errors in my code. Although I faced these challenges, overall, I thought the project was manageable and much simpler than project 3.

Test Cases:

**countMatches():**

1) string d[9] = {

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

};

int i = countMatches(d, 9, "danvers");

int j = countMatches(d, 4, "stark");

int k = countMatches(d, 9, “ro");

Returns 4, 0, 0 as it should.

**detectMatch():**

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

int j = detectMatch(people, 5, "dany");

int k= detectMatch(people, 5, “banner");

int l = detectMatch(people, 0, “banner”);

int m = detectMatch(people, 0, “BAnner");

Returns -1, 3, -1, -1 as it should

**detectSequence():**

1) string d[9] = {

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

};

int b;

int e;

bool b1 = detectSequence(d, 9, "stark", b, e); // returns true and

// sets b to 4 and e to 6

cout << b << e << endl;

bool b2 = detectSequence(d, 9, "romanoff", b, e); // returns true and

// sets b to 1 and e to 1

cout << b << e << endl;

bool b3 = detectSequence(d, 9, "rogers", b, e); // returns false and

// leaves b and e unchanged

bool b4 = detectSequence(d, 0, "rogers", b, e); //returns false

//leaves b and e unchanged

Returns the correct boolean values and sets b and e to the right values.

**detectMin():**

1) string people[6] = { "danvers", "thor", "stark", "banner", "romanoff", "b" };

int j = detectMin(people, 6);

Returns 5 like it should

2) string people[6] = { "danvers", "thor", "Stark", "banner", "romanoff", "b" };

int j = detectMin(people, 6); // returns 3, since banner is earliest in alphabetical order

Returns 2 like it should

3) string people[6] = { "danvers", "thor", "Stark", "banner", "romanoff", "b" };

int j = detectMin(people, 0); // returns 3, since banner is earliest in alphabetical order

Returns -1 like it should

**moveToBack():**

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

int j = moveToBack(people, 5, 0);

Moves danvers to the back, new order is “thor”, “stark”, “banner”, “romanoff”, “danvers”

returns 0 as it should

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

int j = moveToBack(people, 5, 4);

Returns 4 and keeps the array the same order as it should

**moveToFront():**

1) string p[5] = { "danvers", "thor", "stark", "banner", "romanoff" };

int j = moveToFront(p, 5, 1);

Returns 1 and puts thor first as it is supposed to.

2) Same thing, moveToFront(p, 5, 0);

Returns 0 and keeps the array the same as it should

3) Same thing, moveToFront(p, 3, 4)

Returns -1 as it should.

**detectDifference():**

1)string cast[5] = { "danvers", "thor", "barton", "rhodes", "romanoff" };

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

detectDifference(cast, 5, roles, 4);

Returns 4 as it is supposed to.

2) string cast[5] = { "danvers", "thor", "rhodes", "barton", "romanoff" };

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

detectDifference(cast, 5, roles, 3);

Returns 2 as it should.

**deleteDups() :**

1) string d[9] = {"thor", "romanoff", "dan", "danvers", "stark", "thor", "stark", "danvers", “danvers"};

deleteDups(d, 9)

Tests to make sure that even if a string repeats, it will still be counted. Should count everything except for one of the last 2 “danvers”. Returns 8 like it’s supposed to.

2) string d[9] = {"thor", "thor", "thor", "thor", "thor", "thor", "thor", "thor", “thor”};

deleteDups(d, 9)

Tests to make sure that even if all the strings are the same, the function works. Works as planned, returns 1.

**contains():**

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

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

contains(big, 6, little1, 3);

Tests to see if all the strings are there but are in the wrong order, if the function works as intended. It does, returning false.

2) same thing, except I call contains(big, 6, little1, 0);

Returns true as intended.

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

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

contains(big, 9, little1, 4);

Returns true as intended.

**meld():**

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

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

string z[20];

meld(x, 5, y, 4, z, 20);

Returns 9 as expected while also saving the values in z in the correct alphabetical order.

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

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

string z[20];

meld(x, 5, y, 4, z, 20)

Returns -1 as expected

**split():**

1) string g[4] = { "romanoff", "rogers", "rogers", "rogers" };

split(g, 4, “rogers");

Tests to check if split works when there are no strings less than the splitter. Works as planned, returns 0, with “rogers”, “rogers”, “rogers”, “romanoff”

2) string g[7] = { "romanoff", "rogers", "rogers", "rogers", "banner", "danvers", "danvers" };

split(g, 7, “danvers");

Tests with strings less than, equal to and greater than splitter present. Returns 1 as it is supposed to, and ordered correctly, with banner first, followed by both danvers, then the rest of the strings.