a. The biggest obstacle I’ve overcome in this project was writing the contains function. I was confused about how to check if the elements in a2 appear in the same order in a1. I started by writing a nested for loop to check if all elements of a2 are also in a1. Then, I set up another int r and let it be the next of a position at which a match was found. I modified the inner loop and allowed it to only check the elements in a1 starting from the position r. With this modification, the program also checks if the order is correct.

b. string h[7] = { "moana", "tiana", "elsa", "ariel", "", "belle", "elsa" };

string p[6] = { "moana", "tiana", "elsa", "ariel", "", "belle" };

string g[4] = { "moana", "tiana", "ariel", "belle" };

string f[4] = { "belle", "ariel", "tiana", "elsa" };

string q[5] = { "ariel", "belle", "elsa", "tiana", "elsa" };

string e[5] = { "belle", "belle", "belle", "tiana", "tiana" };

string o[7] = { "moana", "tiana", "tiana", "tiana", "tiana", "belle", "elsa" };

string r[7] = { "moana", "moana", "tiana", "tiana", "tiana", "tiana", "belle" };

string s[4] = { "belle", "belle", "belle", "belle"};

string x[4] = { "elsa", "elsa", "raya", "tiana" };

string y[4] = { "ariel", "belle", "elsa", "merida" };

string z[10];

* Looking for the count of matches: assert(countMatches(h, 7, "elsa") == 2);
* If there is no match in the array: assert(countMatches(h, 7, "merida") == 0);
* If there is no element examined: assert(countMatches(h, 0, "elsa") == 0);
* Finding the first occurrence of the match: assert(detectMatch(h, 7, "elsa") == 2);
* If there is no match: assert(detectMatch(h, 2, "elsa") == -1);
* If the value of n is negative: assert(detectMatch(h, -1, "elsa") == -1);

**int** bg;

**int** en;

* Finding the beginning and ending of a sequence of one match: assert(detectSequence(h, 7, "ariel", bg, en) && bg == 3 && en == 3);
* If there is no element examined: assert(detectSequence(h, 0, "ariel", bg, en) == **false**);
* If no match is found: assert(detectSequence(h, 0, "clara", bg, en) == **false**);
* Finding the beginning and ending of a sequence of more than one matches: assert(detectSequence(o, 7, "tiana", bg, en) && bg == 1 && en == 4);
* Finding the position of the smallest element: assert(detectMin(g, 4) == 2);
* If the function should examine no element: assert(detectMin(h, 0) == -1);
* Finding the first element that is different in a1 and a2: assert(detectDifference(h, 4, g, 4) == 2);
* Finding the first element that is different in a1 and a2: assert(detectDifference(f, 4, e, 5) == 1);
* If every examined element is the same in a1 as in a2: assert(detectDifference(h, 7, p, 6) == 6);

* a1 contains all elements of a2 in the same order: assert(contains(h, 7, g, 4));
* n2 is greater than n1: assert(contains(g, 4, h, 7) == **false**);
* a1 contains all elements of a2 but not in the same order: assert(contains(q, 5, f, 4) == **false**);
* Move an element to the last position: assert(moveToBack(g, 4, 1) == 1 && g[1] == "ariel" && g[3] == "tiana");
* Keep an element at the last position: assert(moveToBack(p, 6, 5) == 5 && p[5] == "belle" && p[4] == "");
* Move an element to the last position: assert(moveToBack(p, 6, 3) == 3 && p[3] == "" && p[5] == "ariel");
* Move an element to the first position: assert(moveToFront(f, 4, 2) == 2 && f[0] == "tiana" && f[2] == "ariel");
* Keep an element at the first position: assert(moveToFront(o, 7, 0) == 0 && g[0] == "moana" && o[6] == "elsa");
* Move an element to the first position: assert(moveToFront(o, 7, 5) == 5 && o[5] == "tiana" && o[0] == "belle");
* Delete duplicates of elements and keep one of each: assert(deleteDups(e, 5) == 2 && e[1] == "tiana");
* The function examines no element: assert(deleteDups(e, 0) == 0);
* Delete many duplicates of elements and keep one of each: assert(deleteDups(r, 7) == 3 && r[0] == "moana" && r[1] == "tiana" && r[2] == "belle");
* The size of the result array is smaller than the the size of a1 plus a2: assert(meld(x, 4, y, 4, z, 7) == -1);
* At least one of a1 and a2 is not in non-decreasing order: assert(meld(q, 5, y, 4, z, 9) == -1);
* Combine two arrays in non-decreasing order into the result array, which is also in non-decreasing order: assert(meld(x, 4, y, 4, z, 10) == 8 && z[5] == "merida");
* Finding the first element that is not smaller than the splitter in the rearranged array: assert(split(h, 7, "elsa") == 3);
* Finding the first element that is not smaller than the splitter in the rearranged array: assert(split(r, 7, "elsa") == 1);
* Finding the first element that is not smaller than the splitter in the rearranged array: assert(split(s, 4, "elsa") == 4);