1. When I began to write this program, the first thing I did was write the program so that all of the functions would return -1, and all the Boolean functions would return 0.  
     
   Then I developed the test cases for the first function, and then wrote the first function, developed the test cases for the second function, wrote the second function, etc.  
     
   The first function where I encountered was 4th function, because I did not know how to compare the value of a given entry of the array to all the other entries of the array.

I skipped to the 5th functions and was successful until the 8th function, which was also tricky. I reused the code from moveToEnd in order to overwrite any duplicates I encountered.

When writing subsequence, I made a mistake about checking the first array for entries from the second array. Originally when I wrote the code, it would search the entire first array for the entry of the second array, instead of looking after the entry that it had previously found.

When I wrote positionOfMin, I had to discover that the ~ mark was the largest ASCII character, in order to make sure that the it was the largest string initially, so that the position of the smallest decreasing item was correctly stored.

1. Test cases all 11 functions

**tally:**

z[8] = { "peter", "lois", "meg", "chris", "stewie", "joe" , "joe", "joe" }

tally ( z , 5, “josh”);

should return 0

tally ( z , 8, “joe”);

should return 3

tally ( z , 5, “joe”);

should return 0

tally ( z , -4, “joe”);

should return -1

**findFirst:**

z[8] = { "peter", "lois", "meg", "chris", "stewie", "joe" , "joe", "joe" }

findFirst ( z , 8, “joe”);

should return 5

findFirst ( z , 8, “peter”);

should return 0

findFirst ( z , 8, “meg”);

should return 2

findFirst ( z , 8, “chris”);

should return 3

findFirst ( z , -4, “peter”);

should return -1

findFirst ( z , 8, “michael”);

should return -1

**findFirstSequence:**

a[9] = { "peter", "lois", "meg", "meg", "chris", "stewie", "joe" , "joe", "joe" }

int start = 0, finish = 0;

findFirstSequence( z , 9, "joe", start, finish);

should set start to 6, finish to 8, return 1

findFirstSequence( z , -1, “joe”, start, finish);

should leave start alone, finish alone, return 0

findFirstSequence( z , 9, “meg”, start, finish);

should set start to 2, finish to 3, return 1

**positionOfMin:**

string actors[6] = { "peter", "lois", "meg", "chris", "stewie", "aaron" };

positionOfMin(actors, 5);

returns 3

^^^ given to us

positionOfMin(actors, 3);

returns 1

positionOfMin(actors, -5);

returns -1

positionOfMin(actors, 6);

returns 5

**moveToEnd:**

string actors[5] = { "peter", "lois", "meg", "chris", "stewie" };

moveToEnd(actors, 5, 1);

returns 1 and puts “lois” in the last position and moves the other 3 left one position

^^^ given to us

moveToEnd(actors, 5, 0);

returns 1 and puts “peter” at the end

moveToEnd(actors, -1, 1);

returns -1 and does nothing to actors

**moveToBeginning:**

string actors[5] = { "peter", "lois", "meg", "chris", "stewie" };

moveToBeginning(actors, 5, 2);

returns 2, actors now contains: "meg" "peter" "lois" "chris" "stewie"

^^^ given to us

moveToBeginning(actors, 5, 1);

returns 1, actors now contains: "lois" "peter" "meg" "chris" "stewie"

moveToBeginning(actors, -3, 1);

returns -1, actors unchanged

**disagree:**

string family[5] = { "peter", "lois", "meg", "chris", "stewie" };

string people[6] = { "peter", "lois", "quagmire", "cleveland" };

disagree(family, 5, people, 4);

returns 2

^^^ given to us

disagree(family, 2, people, 1);

returns 1

^^^ given to us

disagree(family, 2, people, 3);

Returns 2;

disagree(family, 3, people, 3);

returns 2;

disagree(family, -1, people, -4);

returns -1;

**removeDups:**

string d[9] = {"lois", "chris", "peter", "peter", "meg", "meg", "meg", "peter", "peter"};

removeDups(d, 9);

returns 5, d is now "lois" "chris" "peter" "meg" "peter"

^^^ given to us

removeDups(d, 5);

returns 4, d is now "lois" "chris" "peter" "meg"

removeDups(d, 3);

returns 3, d is now "lois" "chris" "peter"

removeDups(d, 4);

returns 3, d is now "lois" "chris" "peter"

**subsequence:**

string big[10] = { "peter", "lois", "meg", "chris", "stewie", "meg" };

string little1[10] = { "lois", "chris", "stewie" };

string little2[10] = { "meg", "lois" };

string little3[10] = { "lois", "meg", "meg" };

string little4[10] = { "lois", "lois", "meg" };

subsequence(big, 6, little1, 3);

returns true

subsequence(big, 6, little2, 2);

returns false

subsequence(big, 6, little3, 3);

returns true

subsequence(big, 6, little4, 3);

returns false

^^^ given to us

subsequence(big, 6, little4, 3);

**mingle:**

string x[5] = { "brian", "lois", "meg", "peter", "quagmire" };

string y[4] = { "chris", "lois", "lois", "stewie" };

string z[20];

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

returns 9, z has brian chris lois lois lois meg peter quagmire stewie

^^^ given to us

string q[5] = { "a", "b", "c", "d", "e" };

string w[4] = { "a", "b", "c", "d" };

string e[20];

mingle(q, 5, w, 4, e, 20);

returns 9, e has a a b b c c d d e

**divide:**

string f[6] = { "brian", "meg", "quagmire", "chris", "stewie", "lois" }; string g[4] = { "peter", "meg", "stewie", "lois" };

divide(f, 6, "peter");

returns 4

^^^ given to us

divide(g, 4, "meg");

returns 1

^^^ given to us

divide (f, 5, “peter”);

returns 3