1)

In this project, my greatest difficulties came along with the split function. I couldn’t figure out how to rotate between letters within strings within arrays. After I figured out how to do this, I was confused by how to set the limits of the for loop when rotating through loops of two different sized strings. Eventually I figured this out by setting the size to the smaller of the two strings.

2)

appendToAll

//initial asserts affect following ones

string food[5] = {"pasta", "sushi", "barbeque", "curry", "steak" };

1) assert(appendToAll(food, -6, "!!") == -1 && food[0] == "pasta");

2) //assert(appendToAll(food, 4, "!!") == 4 && food[4] == "steak" && food[3] == "curry!!" );

3) assert(appendToAll(food, 5, "!!") == 5 && food[4] == "steak!!" && food[3] == "curry!!" );

Assert 1 makes sure that the function returns -1 and doesn’t change the array when input - value

Assert 2 makes sure the appending only occurs up to the value of interesting value delineated

Assert 3 makes sure that all of the items in the string are appended correctly

lookUp

string seasons[5] = {"winter", "fall", "spring", "summer", "winter"};

1) assert(lookup(seasons, 5, "winter") == 0);

2) assert(lookup(seasons, 5, "summer") == 3);

3) assert(lookup(seasons, 5, "Summer") == -1);

4) assert(lookup(seasons, 2, "summer") == -1);

5) assert(lookup(seasons, -7, "summer") == -1);

Assert 1 makes sure that looking up a value returns the first correct position

Assert 2 makes sure that looking up a value returns the correct position

Assert 3 makes sure that looking up with capitals is not the same thing and that it returns -1

Assert 4 makes sure that when there is no match in the delineated elements, it returns -1

Assert 5 makes sure that a negative input for n returns -1

positionOfMax

string sports[6] = {"Soccer", "Basketball", "Football", "Rugby", "Tennis", "Track"};

1) assert(positionOfMax(sports, 6) == 5);

2) assert(positionOfMax(sports, -5) == -1);

3) assert(positionOfMax(sports, 4) == 0);

4) assert(positionOfMax(sports, 0) == -1);

Assert 1 makes sure that it returns the position of largest alphabetical value

Assert 2 makes sure that if it is a negative value for n it returns -1

Assert 3 makes sure that it returns the correct position when inspecting only part of array given

Assert 4 makes sure that a input of 0 for n also returns -1

rotateLeft

//initial asserts affect following ones

string trees[6] = {"poplar", "birch", "pine", "oak", "evergreen", "maple"};

1) assert(rotateLeft(trees, -6, 1) == -1 && trees[1] == "birch");

2) assert(rotateLeft(trees, 6, -1) == -1 && trees[1] == "birch");

3) //assert(rotateLeft(trees, 6, 1) == 1 && trees[1] == "pine" && trees[5] == "birch");

4) //assert(rotateLeft(trees, 6, 4) == 4 && trees[1] == "birch" && trees[4] == "maple");

5) assert(rotateLeft(trees, 6, 2) == 2 && trees[1] == "birch" && trees[5] == "pine");

6) assert(rotateLeft(trees, 6, 5) == 5 && trees[1] == "birch" && trees[5] == "maple");

Assert 1 makes sure that a negative input for n causes the function to return -1

Assert 2 makes sure that a negative input for the position also returns -1

Assert 3 makes sure that the function rotates correctly with the right inputs

Assert 4 makes sure that the function rotates correctly with the right inputs

Assert 5 makes sure the function rotates correctly at different positions

Assert 6 makes sure the function rotates correctly if the position given is the last in the array

countRuns

string fruits[5] = {"apple", "apple", "mango", "mango", "orange"};

1) assert(countRuns(fruits, 5) == 3);

2) assert(countRuns(fruits, 4) == 2);

3) assert(countRuns(fruits, 3) == 2);

4) assert(countRuns(fruits, -5) == -1);

5) assert(countRuns(fruits, 1) == 1);

Assert 1 makes sure that the function returns right value with repeats

Assert 2 makes sure that the function returns the correct values with a section of the array

Assert 3 makes sure that the function returns correctly if only a section of the ray has repeats

Assert 4 makes sure that a negative input of n returns -1

Assert 5 makes sure that is the array is only repeats it returns -1

flip

//initial asserts affect following ones

string colors[5] = {"red", "orange", "yellow", "green", "blue"};

1) assert(flip(colors, -3) == -1);

2) //assert(flip(colors, 5) == 5 && colors[0] == "blue" && colors[2] == "yellow");

3) //assert(flip(colors, 4) == 4 && colors[0] == "green" && colors[2] == "orange");

4) assert(flip(colors, 0) == 0 && colors[0] == "red" && colors[2] == "yellow");

Assert 1 checks that a negative input for n returns -1

Assert 2 checks that an odd numbered array flips the sides while the middle value remains

Assert 3 checks that an even numbered numbered array flips along the middle of the array

Assert 4 checks that if the array has no interesting values, it remains the same

differ

string desserts[4] = {"waffle", "pancakes", "cereal", "brownie"};

string breakfast[4] = {"waffle", "pancakes", "cereal", "eggs"};

1) assert(differ(desserts, 4, breakfast, 4) == 3);

2 )assert(differ(desserts, 4, breakfast, 1) == 1);

3) assert(differ(desserts, 4, breakfast,-3) == -1);

4) assert(differ(desserts, -4, breakfast,-3) == -1);

5) assert(differ(desserts, 4, breakfast,-3) == -1);

6) assert(differ(desserts, 3, breakfast, 3) == 3);

Assert 1 checks that if both arrays run through completely, they return correct value

Assert 2 checks that if one array is smaller, it returns the first position where they don’t match

Assert 3 checks that if n2 is negative it returns -1

Assert 4 checks that if n1 and n2 is negative it returns -1

Assert 5 checks that if just n1 is negative it returns -1

Assert 6 checks that if they match all the way through, it returns the value of n1/n2

subsequence

string plant [5] = {"bush", "tree", "flower", "succulent", "fern"};

string plant2 [3] = {"tree", "flower", "succulent"};

string plant3 [3] = {"daisy", "orchid", "lily"};

1) assert(subsequence(desserts, 4, breakfast, 4) == -1);

2) assert(subsequence(plant, 5, plant2, 3) == 1);

3) assert(subsequence(plant, 5, plant2, 0) == 0);

4) assert(subsequence(plant, 5, plant3, 3) == -1);

5) assert(subsequence(plant, -5, plant2, 3) == -1);

6) assert(subsequence(plant, 5, plant2, -3) == -1);

7) assert(subsequence(plant, -5, plant2, -3) == -1);

Assert 1 checks that a partially matching substring still returns -1

Assert 2 checks that a correct substring returns the right position

Assert 3 checks that an empty substring returns 0

Assert 4 checks that a non matching substring returns -1

Assert 5 checks that a negative input for n1 returns -1

Assert 6 checks that a negative input for n2 returns -1

Assert 7 checks that a negative input for n1 and n2 returns -1

lookupAny

string furniture[6] = {"couch", "chair", "table", "TV", "window", "pot"};

string furniture2[4] = {"desk", "window", "curtain", "table"};

1) assert(lookupAny(furniture, 6, furniture2, 4) == 2);

2) assert(lookupAny(furniture, 6, furniture2, 1) == -1);

3) assert(lookupAny(furniture, -6, furniture2, 4) == -1);

4) assert(lookupAny(furniture, 6, furniture2, -4) == -1);

5) assert(lookupAny(furniture, -6, furniture2, -4) == -1);

Assert 1 checks that if both strings are both run through completely it gives right output

Assert 2 checks that if no matches are found it returns -1

Assert 3 checks that if n1 is negative it returns -1

Assert 4 checks that if n2 is negative it returns -1

Assert 5 checks that if n1 and n2 is negative it returns -1

split

//initial asserts affect following ones

string dogs[5] = {"Husky", "Bulldog", "Boxer", "Poodle", "Labrador"};

1) assert(split(dogs, 5, "liz") == 4 );

2) assert(split(dogs, 5, "couch") == 2 && ("bulldog" == dogs[0] || "bulldog" == dogs[1]));

3) assert(split(dogs, 5, "quartz") == 5);

4) assert(split(dogs, -5, "couch") == -1);

5) assert(split(dogs, 5, "abc") == 0);

Assert 1 makes sure that the value of splitter returns correctly

Assert 2 checks that strings in the array are shifted according to the splitter

Assert 3 checks that the function returns n if there are no such values that >splitter

Assert 4 checks that if n is negative, it returns -1

Assert 5 checks that if all values >splitter, the function returns 0 and is unchanged