**Obstacles**

Thinking about what n = 0 should result in

* Hard to imagine how an empty array would react logically in the function
* Had to see if my function had accounted for it or if I had to write a separate if (n == 0) return 0 segment of code

positionOfMax

* Had to think about how to make sure that only the smallest position of two equal maximum string values would be returned

rotateLeft

* Had to make sure that comparing element k with k+1 would not try to go out of the array
* Made condition k+1 < n

flip

* Took a while to figure out to use k < n/2, and to just swap the first half of the array with the second half without actually letting the loop run to the second half of the array
* Had to think about how to make sure the middle element if n was odd would not change place, used int features (where 2.5 becomes 2 instead of 3)

subsequence

* Many nesting loops
* Had to think for a while to come up with idea to compare number n2 with number of matches between a2 and section in a1 (refer to count++ in code)

split

* Had to think for a while to account for situations when the splitter was in the array, and when it wasn’t
* Easier when splitter isn’t present
* When splitter is present, have to store element in splitter’s position, and then replace that element with splitter, and then find the spot where the splitter initially was and replace with this element

**Test cases**

int appendToAll(string a[], int n, string value)

string a[6] = { "bernie", "hillary", "donald", "marco", "carly", "ben" };

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| int n | string value | Reason | Expected a[1] | Expected result |
| -1 | ? | int n is negative | hillary | -1 |
| 2 | ? | correct scenario | hillary? | 2 |
| 0 | ? | int n is 0 | hillary | 0 |

int lookup(const string a[], int n, string target)

string a[6] = { "bob", "hillary", "bernie", "marco", "carly", "ben" };

string b[6] = { "bernie", "hillary", "bernie", "marco", "carly", "ben" };

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | int n | string target | Reason | Expected result |
| string a[] | 6 | "bernie" | One target in array | 2 |
| string b[] | 6 | "bernie" | Two targets in array | 0 |
| string a[] | 6 | "adam" | Target not in array | -1 |
| string a[] | -5 | "bernie" | int n is negative | -1 |
| string a[] | 0 | "bernie" | int n is 0 | -1 |

int positionOfMax(const string a[], int n)

string a[7] = { "bob", "hillary", "bernie", "jeb", "", "carly", "ben" };

string b[7] = { "bernie", "jeb", "bernie", "jeb", "", "carly", "ben" };

|  |  |  |  |
| --- | --- | --- | --- |
|  | int n | Reason | Expected result |
| string a[] | 0 | Array has no elements | -1 |
| string a[] | 7 | Correct scenario | 3 |
| string a[] | 3 | Correct scenario but with smaller array | 1 |
| string b[] | 7 | Maximum value string occurs twice | 1 |
| string a[] | -3 | int n is negative | -1 |
| string a[] | 0 | int n is 0 | -1 |

int rotateLeft(string a[], int n, int pos)

string a[4] = { "bob", "hillary", "bernie", "jeb" };

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| int n | int pos | Reason | Expected a[0] | Expected a[1] | Expected a[2] | Expected a[3] | Expected result |
| 4 | 1 | Correct scenario | "bob" | "bernie" | "jeb" | "hillary" | 1 |
| 3 | 1 | Correct scenario but smaller section of array | "bob" | "bernie" | "hillary" | "jeb" | 1 |
| -3 | 1 | int n is negative | "bob" | "hillary" | "bernie" | "jeb" | -1 |
| 4 | -3 | int pos is negative | "bob" | "hillary" | "bernie" | "jeb" | -1 |
| 0 | 3 | int n is 0 | "bob" | "hillary" | "bernie" | "jeb" | -1 |

int countRuns(const string a[], int n)

string a[7] = { "hillary", "hillary", "hillary", "ben", "ben", "joe", "joe" };

|  |  |  |
| --- | --- | --- |
| int n | Reason | Expected result |
| 7 | Correct scenario with full array | 3 |
| 6 | Correct scenario with section of array | 3 |
| 4 | Correct scenario with smaller section of array | 2 |
| -3 | int n is negative | -1 |
| 0 | int n is 0 | 0 |

int flip(string a[], int n)

string a[4] = { "bob", "hillary", "bernie", "jeb" };

string b[3] = { "bernie", "jeb", "bob" };

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | int n | Reason | Expected a[0] | Expected a[1] | Expected a[2] | Expected a[3] | Expected result |
| string a[] | 4 | Even number of elements | "jeb" | "bernie" | "hillary" | "bob" | 4 |
| string a[] | 3 | Section of array | "bernie" | "hillary" | "bob" | "jeb" | 3 |
| string b[] | 3 | Odd number of elements | "bob" | "jeb" | "bernie" | NA | 3 |
| string b[] | -2 | int n is negative | "bob" | "hillary" | "bernie" | "jeb" | -1 |
| string a[] | 0 | int n is 0 | "bob" | "hillary" | "bernie" | "jeb" | 0 |

int differ(const string a1[], int n1, const string a2[], int n2)

string a[7] = { "bernie", "jeb", "bernie", "bob", "", "carly", "ben" };

string b[7] = { "bernie", "jeb", "bernie", "jeb", "", "carly", "ben" };

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| const string a1[] | int n1 | const string a2[] | int n2 | Reason | Expected result |
| string a[7] | 7 | string b[7] | 7 | Comparing arrays large enough not to run out of elements before reaching difference | 3 |
| string a[7] | 7 | string b[7] | 2 | Comparing two a smaller array where elements run out before reaching difference | 2 |
| string a[7] | 5 | string b[7] | -1 | int n2 is negative | -1 |
| string a[7] | -1 | string b[7] | 5 | int n1 is negative | -1 |
| string a[7] | 0 | string b[7] | 5 | int n1 is 0 | -1 |
| string a[7] | 5 | string b[7] | 0 | int n1 is 0 | -1 |

int subsequence(const string a1[], int n1, const string a2[], int n2)

string a[7] = { "bernie", "jeb", "bernie", "bob", "", "carly", "ben" };

string b[4] = { "jeb", "bernie", "bob", "" };

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| const string a1[] | int n1 | const string a2[] | int n2 | Reason | Expected result |
| string a[7] | 7 | string b[4] | 4 | Comparing arrays large enough not to run out of elements before reaching difference | 1 |
| string a[7] | 7 | string b[4] | 2 | Comparing two a smaller array where elements run out before reaching difference | 1 |
| string a[7] | 0 | string b[4] | 0 | int n1 and n2 is 0 | 0 |
| string a[7] | 5 | string b[4] | 0 | int n2 is 0 | 0 |
| string a[7] | -2 | string b[4] | 4 | int n1 is negative | -1 |
| string a[7] | 5 | string b[4] | -5 | int n2 is negative | -1 |
| string a[7] | 3 | string b[4] | 4 | int n1 is smaller than int 2 | -1 |

int lookupAny(const string a1[], int n1, const string a2[], int n2)

string a[7] = { "adam", "jeb", "bernie", "bob", "", "carly", "ben" };

string b[3] = { "ben", "carly", "bob" };

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| const string a1[] | int n1 | const string a2[] | int n2 | Reason | Expected result |
| string a[7] | 7 | string b[3] | 3 | Correct scenario | 3 |
| string a[7] | 7 | string b[3] | 2 | Another correct scenario | 5 |
| string a[7] | 3 | string b[3] | 2 | a1 does not contain any elements in a2 | -1 |
| string a[7] | -2 | string b[4] | 3 | int n1 is negative | -1 |
| string a[7] | 5 | string b[4] | -2 | int n2 is negative | -1 |
| string a[7] | 0 | string b[4] | 3 | int n1 is 0 | -1 |
| string a[7] | 5 | string b[4] | 0 | int n2 is 0 | -1 |

int split(string a[], int n, string splitter)

string a[7] = { "adam", "jeb", "hillary", "bob", "carly", "ben" };

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| int n | string splitter | Reason | Expected  a[0], a[1], a[2], … | Expected result |
| 6 | "carly" | Splitter in array | **adam, bob, ben, carly, jeb, hillary** | 3 |
| 6 | "carl" | Splitter not in array | **adam, bob, ben, jeb, carly, hillary** | 3 |
| 6 | "zarl" | Splitter is bigger than all elements | **adam, jeb, hillary, bob, carly, ben** | 6 |
| 6 | "aarl" | Splitter is smaller than all elements | **adam, jeb, hillary, bob, carly, ben** | 0 |
| -4 | “carly” | int n is negative | **adam, jeb, hillary, bob, carly, ben** | -1 |
| 0 | “carly” | int n is 0 | **adam, jeb, hillary, bob, carly, ben** | 0 |