Artiom Arutiunov

ID: 504597668

Professor Stahl

Project 4 Report

a. There were several obstacles I encountered while working on this project. The biggest challenge I ran into was figuring out how to keep one element of an array constant while comparing it to an entire other array. Once I realized how to do this, the second most difficult task was how to check for duplicates in the functions that do not allow them. For example, in the function unionWithNoDuplicates, it wasn’t obvious to me that I had to initialize a variable within the first “for” loop that I could then use to check for duplicates. The last obstacle I ran into was figuring out how to not change the value of the minimum variable when comparing an array’s elements against itself. After I realized that I had to use two “for” loops to make sure that the variable wasn’t altered, then the function worked perfectly.

b. Test Cases

After writing all the functions, I first tested my code against the set of tests Howard provided in the spec:

string a[6] = { "alpha", "beta", "gamma", "gamma", "beta", "delta" };

string b[6] = { "delta", "gamma", "beta", "alpha", "beta", "alpha" };

string c [5] = {"cat", "hi", "hell", "hello", "heroes"};

string d[7] = { "a" , "a" , "a" , "a", "aaa", "a", "aaaaaaa"};

string h[7] = { "samwell", "jon", "margaery", "daenerys", "", "tyrion", "margaery" };

int answer;

string output[6] = { " ", " ", " ", " ", " ", " " };

int outputSize = 6;

assert(locateMinimum(a, 3 ) == 0);

assert(locateMinimum(b, 3 ) == 2);

assert(locateMinimum(d, 7 ) == 0);

🡪 Doing some standard testing for the function locateMinimum, the compiler agreed with both of these assertions.

assert(findLastOccurrence(a, 5, "beta" ) == 4);

assert(findLastOccurrence(b, 3, "beta" ) == 2);

assert (findLastOccurrence(b, 6, "beta") == 4);

assert(findLastOccurrence(d, 7, "a" ) == 5);

assert(findLastOccurrence(d, 7, "abcdefg" ) == -1);

assert(findLastOccurrence(d, 7, "aaa" ) == 4);

🡪 Doing some standard testing for the function findLastOccurence, the compiler agreed with both of these assertions. Note that the last assertion makes sure that the function returns the *last* occurrence of “beta.”

assert(hasNoDuplicates(a, 3) == true);

assert(hasNoDuplicates(a, 6) == false);

assert(hasNoDuplicates(b, 6) == false);

assert(hasNoDuplicates(d, 7) == false);

assert(hasNoDuplicates(c, 5) == true);

assert(hasNoDuplicates(d, 2) == false);

🡪 Doing some standard testing for the function hasNoDuplicates, the compiler agreed with both of these assertions.

assert(isInIncreasingOrder(a, 3) == true);

assert(isInIncreasingOrder(a, 6) == false);

assert(isInIncreasingOrder(a, 1) == true);

assert(isInIncreasingOrder(a, 0) == true);

assert(isInIncreasingOrder(a, -1) == false);

assert(isInIncreasingOrder(d, 7) == false);

🡪 Doing some standard testing for the function has isInIncreasingOrder, the compiler agreed with both of these assertions.

unionWithNoDuplicates(a, 6, b, 2, output, outputSize );

assert( outputSize == 4 );

assert( output[0] == "alpha" );

🡪 The compiler agreed with the assertion that the first element of array output was “alpha.”

answer = shiftRight(h, 6, 1, "foobar" );

assert( answer == 5 );

assert( h[0] == "foobar" );

🡪 The compiler agreed with the assertion that there are still 5 elements left in the array after the shift, and that the placeholder string “footbar” occupies the 0th position.

assert(flipAround(b, 3 ) == 1);

assert(flipAround(b, 4 ) == 2);

assert(flipAround(b, 2 ) == 1);

assert(flipAround(d, 7 ) == 3);

🡪 Doing some basic testing, the compiler agreed with these assertions.

Next, I came up with my own test cases to run my code against some more challenging situations.

test cases for locateMinimum:

string a [5] = {"aaa", "aa", "a", "aaaa", "a"};

string b [7] = {"aaa", "bbb", "aa", "bbbb", "a", "bb", "a"};

assert (locateMinimum(a, 5) == 2);

assert (locateMinimum (b, 7) == 4);

🡪 Compiler agreed with both of these assertions. These test cases made sure the function returned the smallest index value of the minimum if it had duplicates.

test cases for findLastOccurence:

string c [5] = {"alpha", "beta", "gamma", "delta", "beta"};

string d [5] = {"alpha", "beta", "delta", "delta", "epsilon"};

string e [1] = {" "};

assert (findLastOccurrence ( c, 5, "beta") == 4);

assert (findLastOccurrence (d, 5, "delta") == 3);

assert (findLastOccurrence(d, 5, "gamma") == -1);

assert (findLastOccurrence(e, 1, "a") == -1);

🡪 Compiler agreed with both of these assertions. These test cases made sure the function returned a value of -1 if the element was not found and the last occurrence of an element if it was found in the array multiple times.

test cases for flipAround:

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

string g [10] = {"z", "x", "a", "a", "a", "z", "e", "f", "eight", "z"};

assert (flipAround (f, 5) == 2);

assert (flipAround (g, 10) == 5);

assert (flipAround (f, 0) == 0);

🡪 Compiler agreed with both of these assertions, testing the code to make sure that it could handle any kind of strings as well as sizes, as well as if none of the elements were flipped around.

test cases for hasNoDuplicates:

string h [6] = {"alpha", "beta", "gamma", "delta", "epsilon"};

string i [1] = {" "};

string j [6] = {"alpha", "beta", "alpha", "beta", "gamma", "alpha"};

string k [0];

assert ( hasNoDuplicates (h, 5));

assert ( hasNoDuplicates (i, 1));

assert (! hasNoDuplicates (j, 6));

assert ( hasNoDuplicates(k, 0));

🡪 Compiler agreed with these assertions, testing for duplicates within the array as well as an array of size 1 and an array of size 0.

test cases for unionWithNoDuplicates:

string l [5] = {"a", "b", "a", "d", "a"};

string m [5] = {"e", "e", "e", "d", "b"};

string n [5] = {"e", "e", "b", "d", "z"};

string o [3] = {"a", "b", "c"};

string p [2] = {"e", "f"};

string resultingString[20];

int resultingSize = 0;

unionWithNoDuplicates(l, 5, m, 5, resultingString, resultingSize);

for (int k = 0; k < resultingSize; k++)

cout << resultingString[k] << endl;

cout << "---" << endl;

unionWithNoDuplicates(m, 5, n, 5, resultingString, resultingSize);

for (int j = 0; j < resultingSize; j++)

cout << resultingString[j] << endl;

cout << "---" << endl;

unionWithNoDuplicates(l, 5, n, 5, resultingString, resultingSize);

for (int k = 0; k < resultingSize; k++)

cout << resultingString[k] << endl;

cout << "---" << endl;

unionWithNoDuplicates(o, 3, p, 2, resultingString, resultingSize);

for (int z = 0; z < resultingSize; z++)

cout << resultingString[z] << endl;

cout << "---" << endl;

unionWithNoDuplicates(l, 5, m, -1, resultingString, resultingSize);

🡪 I tested for unionWithNoDuplicates in a slightly different fashion, using a small “for” loop to check for the actual contents of the array and then number of elements the array has. Additionally, I gave the program an invalid input of a string size less than 0, and the function assigned -1 to the size of resultingSize.

test cases for shiftRight:

string q [10] = {"a", "b", "c", "d", "e", "f", "g", "h", "i", "j"};

shiftRight (q, 10, 4, "$");

for (int i = 0; i < 10; i++)

cout << q[i] << endl;

🡪 Similarly to unionWithNoDuplicates, I made sure that the resulting array had the correct amount of “$” in it and returned the proper amount of original elements left.

test cases for isInIncreasingOrder:

string s [5] = {"a", "aa", "aaa", "aaaa", "aaaaa"};

string t [6] = {"xy", "xy", "x", "x", "y", "xy"};

string r [5] = {"hi", "cat", "hell", "hello", "heroes"};

assert (isInIncreasingOrder (s, 5));

assert (! isInIncreasingOrder(t, 6));

assert (! isInIncreasingOrder (r, 5 ));

🡪 The compiler agreed with these assertions, making sure that it correctly discerns between different strings (in terms of memory allocation, not string size).