Raj Kumar

Professor Smallberg

Computer Science 31

12 November 2020

Project 4 Report

**Obstacles:**

The largest obstacle was understanding what the n=0 inputs would do in each function. For some of the functions, I basically just guessed what would make sense, while others I was sure I was treating the n=0 inputs correctly.

My second obstacle was avoiding fencepost type errors in my sorting technique for the divide function, which becomes significantly easier with a sorted array of strings. It forced me to follow the code closely and use debugging tools on visual studio to help me watch each variable change step by step to find my errors.

My third obstacle was figuring out ways to properly test my functions. There was no Linux Server tester this time, so I had relatively less faith in my code. Fortunately, I was able to make very in-depth testers using assert in my main method. I helped myself catch my mistakes by making a printing function that printed my arrays so I could see how functions like flip changed the arrays inputted.

It is a lot less confusing to just read the code and assert statements than to explain the inputs in this case, so I have explained why I used a test case and then showed the code for it. There were also some cases provided by Professor Smallberg, but I will not be explaining these as I just added them in because they were definitely correct and were a good check to the accuracy of my code.

Each section of testers is headed by a big bolded comment which says which function’s testers are after it, and smaller bolded comments apply as a common reason of testing to the next few lines of code. If you are in a hurry to see my explanations, read the green comments. Some of the comments say why the test works, and for those, I am testing those to see if the functions match what I believe should happen if the code was correct (essentially checking for correctness).

**Test Data: (**All test cases were handled in a way the spec requires)

//TESTS FOR EACH FUNCTION, ONE BY ONE

**//appendToAll tests**

cerr << endl << "appendToAll tests" << endl;

string people[5] = { "donald", "joe", "mike", "lindsey", "kamala" };

assert(appendToAll(people, 5, "!!!") == 5); // returns 5

// now people[0] is "donald!!!", people[1] is "joe!!!", ...,

// and people[4] is "kamala!!!"

assert(appendToAll(people, -1, "!!!") == -1); //returns -1, since you can't have negative array length

assert(appendToAll(people, 0, "!!!") == 0);//checks if returns 0 because this changes nothing and n=0

printArray(people, 5); //to see if array actually changed or not

**//positionOfMax tests**

cerr << endl << "positionOfMax tests" << endl;

string candidate[6] = { "jamie", "lindsey", "mark", "susan", "joe", "donald" };

string d7[5] = { "mark", "mark", "mark", "susan", "susan" };

assert(positionOfMax(candidate, 6) == 3); // returns 3, since susan is latest in alphabetic order

assert(positionOfMax(candidate, 3) == 2); //returns 2, since mark is latest in alphabetical order

assert(positionOfMax(candidate, 0) == -1); //returns -1, since you can't find a max in nothing

assert(positionOfMax(candidate, -1) == -1); //returns -1, since you can't have negative array length

assert(positionOfMax(d7, 5) == 3); //returns 3, since susan is latest in alphabetical order and appears first at index 3

**//rotateLeft tests**

cerr << endl << "rotateLeft tests" << endl;

string politician[5] = { "kamala", "jamie", "lindsey", "sara", "mark" };

string candidate123[6] = { "jamie", "lindsey", "mark", "susan", "joe", "donald" };

assert(rotateLeft(politician, 5, 1) == 1); // returns 1 to check if the correct result of 1 is returned, to check if it just works

// politician now contains: "kamala", "lindsey", "sara", "mark", "jamie"

assert(rotateLeft(candidate123, 6, 3) == 3); //to check if the correct result of 3 is returned, to check if it just works

assert(rotateLeft(candidate123, 6, 7) == -1); //can't look out of bounds

assert(rotateLeft(candidate123, 6, -1) == -1); //can't look out of bounds

assert(rotateLeft(politician, 0, 1) == -1); //n is 0 so return -1, said in FAQ #4

**//countRuns tests**

cerr << endl << "countRuns tests" << endl;

string d2[9] = {"susan", "donald", "mike", "mike", "joe", "joe", "joe", "mike", "mike"};

assert(countRuns(d2, 9) == 5); // returns 5

// The five sequences of consecutive identical items are

// "susan"

// "donald"

// "mike", "mike"

// "joe", "joe", "joe"

// "mike", "mike"

assert(countRuns(d2, -1) == -1); //-1 because you can't have negative number of elements

assert(countRuns(d2, 0) == 0); //0 runs in empty array, check with smallberg on monday

assert(countRuns(d2, 1) == 1); //one run if only one element

assert(countRuns(d7, 5) == 2); //mark is first run, susan is second run

**//flip tests**

cerr << endl << "flip tests" << endl;

string folks[6] = { "donald", "mike", "", "susan", "sara", "jamie" };

assert(flip(folks, 4) == 4); // returns 4

// folks now contains: "susan" "" "mike" "donald" "sara" "jamie"

assert(flip(d2, 8) == 8); // see if this flips a longer string correctly or not

assert(flip(d2, 1) == 1); //see if this is the same or not

assert(flip(folks, 0) == 0); //makes no sense to flip an empty array

**//differ tests**

cerr << endl << "differ tests" << endl;

string folks2[6] = { "donald", "mike", "", "susan", "sara", "jamie" };

string folks21[6] = { "donald", "mike", "", "susan", "sara", "jamie" };

string folks22[7] = { "donald", "mike", "", "susan", "sara", "jamie", "kungpaochicken"};

string group[5] = { "donald", "mike", "jamie", "", "susan" };

**//to check if program just works or not in regular cases**

assert(differ(folks2, 6, group, 5) == 2); // returns 2

assert(differ(folks2, 2, group, 1) == 1); // returns 1

assert(differ(folks2, 6, folks21, 6) == 6); // returns 1

assert(differ(folks2, 6, folks22, 7) == 6); // returns 6

**//to check if program understands that an input of n1 or n2 being 0 will mean that they differ from the very start, the 0th index**

assert(differ(folks2, 6, folks22, 0) == 0); // returns 0

assert(differ(folks2, 0, folks22, 6) == 0); // returns 0

assert(differ(folks2, 0, folks22, 0) == 0); // returns 0

**//to check if function recognizes these are invalid inputs, can't have negative array length**

assert(differ(folks2, 6, folks22, -2) == -1); // returns -1

assert(differ(folks2, -2, folks22, 3) == -1); // returns -1

**//to check if n1 or n2 is returned (whichever is shorter) if reached end of either string**

assert(differ(folks2, 6, folks2, 6) == 6); // returns 6

**//subsequence tests**

cerr << endl << "subsequence tests" << endl;

string names[10] = { "kamala", "mark", "sara", "martha", "donald", "lindsey" };

string names1[10] = { "mark", "sara", "martha" };

string names2[10] = { "kamala", "martha" };

string names3[1] = { "lindsey" };

string names4[10] = {"sara", "martha", "donald" };

assert(subsequence(names, 6, names1, 3) == 1); // returns 1 b/c mark is at index 1 in names

assert(subsequence(folks2, 5, folks21, 2) == 0); // returns 0 to check what happens if second string array is part of first string array from very start (index 0)

assert(subsequence(folks2, 5, folks2, 6) == -1); // returns -1 b/c n2 can't fit inside n1 and function recognizes this

assert(subsequence(names, 6, names4, 3) == 2); // returns 2 b/c sara is at index 2 in names

assert(subsequence(names, 6, names3, 1) == 5); // returns 5 b/c lindsey is at index 5 in names

**//lookupAny tests**

cerr << endl << "lookupAny tests" << endl;

string set1[10] = { "jamie", "donald", "martha", "mark" };

**//to see if regular test cases work as expected**

assert(lookupAny(names, 6, set1, 4) == 1); // returns 1 (a1 has "mark" there)

assert(lookupAny(names, 6, folks2, 5) == 2); // returns 2 (a1 has "sara" there)

assert(lookupAny(names, 6, d2, 9) == 4); // returns 1 (a1 has "mark" there)

string set2[10] = { "susan", "joe" };

assert(lookupAny(names, 6, set2, 2) == -1); // returns -1 to check if function can see that there is no matches

**// to see if having 0 length n1 or n2 is correctly identified as -1, which shows there can't be a match between the two arrays if one is empty**

assert(lookupAny(names, 6, set2, 0) == -1); // returns -1

assert(lookupAny(names, 0, set2, 6) == -1); // returns -1

**//divide tests**

cerr << endl << "divide tests" << endl;

string candidate45[6] = { "jamie", "lindsey", "mark", "susan", "joe", "donald" };

assert(divide(candidate45, 6, "kamala") == 3); // returns 3

// candidate must now be

// "jamie" "joe" "donald" "lindsey" "susan" "mark"

// or "joe" "donald" "jamie" "mark" "lindsey" "susan"

// or one of several other orderings.

// All elements < "kamala" (i.e., "joe", "donald", and "jamie")

// come before all others

// All elements > "kamala" (i.e., "susan", "lindsey", and "mark")

// come after all others

string candidate56[4] = { "mark", "sara", "lindsey", "mike" };

assert(divide(candidate56, 4, "mike") == 2); // returns 2

// candidate2 must now be either

// "mark" "lindsey" "mike" "sara"

// or "lindsey" "mark" "mike" "sara"

// All elements < "mike" (i.e., "lindsey" and "mark") come

// before all others.

// All elements > "mike" (i.e., "sara") come after all others.

string candidate1234[1] = { "mark" };

string candidate12345[3] = { "mark", "mike", "pence" };

**//to check if regular cases work or not**

assert(divide(candidate1234, 1, "mike") == 1); // returns 1 b/c mike is greater than the one string in candidate 1234

assert(divide(candidate1234, 1, "a") == 0); // returns 0 b/c index 0 works

//odd or edge cases (explanation why is given)

assert(divide(candidate, 6, "z") == 6); // returns 6, which is n, to check what happens when z is greater than all strings in candidate

assert(divide(candidate12345, 3, "a") == 0); // returns 0, to check what happens when a is less than all strings in candidate12345