Com Sci 31 – Winter 2019

Project 4 Report

Name – Suyash Kumar

UID – 605108040

Discussion – 1C

# Notable obstacles overcame

* Paying careful attention to project specifications was necessary to avoid minute errors.
* I followed incremental development by developing each function one at a time and thoroughly testing for good as well as bad cases.
* Creating the shiftRight() function was somewhat tricky. However, through careful consideration of logic as well as my main objective, I was able to complete it.
* I tried to avoid writing unnecessary code wherever possible to keep my code compact, clean, and comprehensive.
* I have also included comments wherever necessary to make clear what I was intending to achieve with my code, in cases where it may not be immediately obvious (like in the shiftRight() function).

# Test cases

Test cases have been sorted by function, and within each function there are instances of good as well as bad test cases.

hasDuplicates() function

Good Test Cases (Having legal size values > 0)

1. Input array: string subjects[5] = {“physics”,”chemistry”,”cs”,”physics”,”math”};

Call statement: cout<<hasDuplicates(subjects,5)<<endl;

Output: true

1. Input array: string testarr[3] = {“”,”middle”,””}; (works with empty strings)

Call statement: cout<<hasDuplicates(testarr,3)<<endl;

Output: true

1. Input array: string persons[4] = {“tony”,”helen”,”david”,”jared”};

Call statement: cout<<hasDuplicates(persons,3)<<endl; (works with a size parameter less than the actual array size)

Output: false

Bad Test Cases (Having illegal size values)

1. Input array: string classes[3] = {“english”,”cs”,”russian”};

Call statement: cout<<hasDuplicates(classes,-10)<<endl;

Output: false

countAllDigits() function

Good Test Cases (Having legal size values > 0)

1. Input array: string array1[4] = {“123abc”,”maxwell”,”kr12”,”goodnight”};

Call statement: cout<<countAllDigits(array1,4)<<endl;

Output: 5

1. Input array: string array2[3] = {“1234”,””,””); (works with empty strings)

Call statement: cout<<countAllDigits(array2,2)<<endl;

Output: 4

1. Input array: string array3[4] = {“abc”,”maxwell”,”kr”,”goodnight”};

Call statement: cout<<countAllDigits(array1,4)<<endl;

Output: 0

Bad Test Cases (Having illegal size values)

1. Input array: string array0[2] = {“avast”,”mech12” };

Call statement: cout<<countAllDigits(array1,0)<<endl;

Output: -1

isInDecreasingOrder() function

Good Test Cases (Having legal size values >= 0)

1. Input array: string desc[3] = {“Carl”,”Beta”,”Anderson”};

Call statement: cout<<isInDecreasingOrder(desc,3)<<endl;

Output: true

1. Input array: string nondesc[4] = {“Carl”,”Beta”,”Anderson”,”andy”};

Call statement: cout<<isInDecreasingOrder(nondesc,4)<<endl;

Output: false

1. Input array: string single[1] = {“Hello”}; (returns true for an array of size 1)

Call statement: cout<<isInDecreasingOrder(single,1)<<endl;

Output: true

1. Input array: string empty[0]; (returns true for an array of size 0)

Call statement: cout<<isInDecreasingOrder(empty,0)<<endl;

Output: true

Bad Test Cases (Having illegal size values)

1. Input array: string bad[4] = {“Hello”,”ByeBye”,””,”123”};

Call statement: cout<<isInDecreasingOrder(bad,-10)<<endl;

Output: false

shiftRight() function

Good Test Cases (Having legal values of size >=0 and amount>=0)

1. Input array: string sample[5] = {“max”,”decker”,”rose”,”white”,”violet”};

Call statement: cout<<shiftRight(sample,5,2,”hello”)<<endl;

Output: 2 (sample becomes {“hello”,”hello”,”max”,”decker”,”rose”})

1. Input array: string sample[5] = {“max”,”decker”,”rose”,”white”,”violet”};

Call statement: cout<<shiftRight(sample,3,2,”hello”)<<endl;

Output: 2 (sample becomes {“hello”,”hello”,”max”,”white”,”violet”}, the last two elements of the array are not accessed at all)

1. Input array: string sample1[3] = {“good”,”okay”,”bad”};

Call statement: cout<<shiftRight(sample1,3,10,”foo”)<<endl;

Output: 10 (sample1 becomes {“foo”,”foo”,”foo”})

1. Input array: string sample[5] = {“max”,”decker”,”rose”,”white”,”violet”};

Call statement: cout<<shiftRight(sample,5,0,”hello”)<<endl;

Output: 0 (sample remains the same)

1. Input array: string sample[5] = {“max”,”decker”,”rose”,”white”,”violet”};

Call statement: cout<<shiftRight(sample,0,2,”hello”)<<endl;

Output: 0 (sample remains the same)

1. Input array: string sample1[3] = {“good”,”okay”,”bad”};

Call statement: cout<<shiftRight(sample1,0,0,”foo”)<<endl;

Output: 0 (sample1 remains the same)

Bad Test Cases (Having illegal size and/or amount values)

1. Input array: string badarr[2] = {“physics”,”math”};

Call statement: cout<<shiftRight(badarr,2,-4,”foo1”)<<endl;

Output: -1 (badarr remains the same)

1. Input array: string badarr[2] = {“physics”,”math”};

Call statement: cout<<shiftRight(badarr,0,-4,”foo1”)<<endl;

Output: -1 (badarr remains the same)

1. Input array: string badarr[2] = {“physics”,”math”};

Call statement: cout<<shiftRight(badarr,-4,0,”foo1”)<<endl;

Output: -1 (badarr remains the same)

1. Input array: string badarr[2] = {“physics”,”math”};

Call statement: cout<<shiftRight(badarr,-10,-4,”foo1”)<<endl;

Output: -1 (badarr remains the same)

find() function

Good Test Cases (Having legal size values > 0)

1. Input array: string new[4] = {“sunshine”,”hope”,”love”,”peace”};

Call statement: cout<<find(new,4,”hope”)<<endl;

Output: 1

1. Input array: string old[2] = {“Germany”,”Russia”};

Call statement: cout<<find(old,2,”USA”)<<endl;

Output: -1 (as “USA” was not found)

Bad Test Cases (Having illegal size values)

1. Input array: string errorcase[3] = {“Error1”,”Error2”,”Error3”};

Call statement: cout<<find(errorcase,-10,”Error1”)<<endl;

Output: -1

replaceAll() function

Good Test Cases (Having legal size values > 0)

1. Input array: string replace[3] = {“sam”,”alex”,”peter”};

Call statement: cout<<replaceAll(replace, 3, ‘a’, ‘\*’)<<endl;

Output: 2 (replace becomes {“s\*m”,”\*lex”,”peter”})

1. Input array: string new[2] = {“hhh”,”qqq”};

Call statement: cout<<replaceAll(new, 2, ‘a’, ‘0’)<<endl;

Output: 0 (new remains the same)

Bad Test Cases (Having illegal size values)

1. Input array: string finalerror[2] = {“hello”,”welcome”};

Call statement: cout<<replaceAll(finalerror,-12,’e’,’x’)<<endl;

Output: -1