Wenjie Mo

Professor Smallberg

CS 31

5 Nov 2019

Project 4 Report

1. Main obstacles
   1. Infinite loop and the loop will go outside the array.

Solution: use the debugger to find the value of variables and change the operation.

* 1. Several test cases missed commas at first!!! (Thanks for correcting that). So I spent a lot of time checking whether my function itself has problems :( .

1. Test cases

string a[6] = { "zero", "one", "two", "three", "four", "five"};

appendToAll(a, -2, "???") //n is negative, return -1

appendToAll(a, 0, "???")// n is 0, return 0

appendToAll(a, 6, "???") //normal case

string bb[6] = { "zero", "one", "two", "three", "four", "five"};

lookup(bb, -2, "one") // n is negative, return -1

lookup(bb, 2, "zero") // normal case

lookup(bb, 6, "seven") // normal case: no match

string cc[6] = { "zero", "one", "two", "three", "four", "five"};

positionOfMax(cc, 0) // n is negative or 0, there is no max, return -1

positionOfMax(cc, 1) // n is 1, there is no max, return the first one

positionOfMax(cc, 6) // normal case, zero is the max

string dd[6] = { "zero", "one", "two", "three", "four", "five"};

rotateLeft(dd, -2, 0) // n is negative

rotateLeft(dd, 6, 2) // normal case

string ee[6] = { "zero", "one", "one", "two", "two", "two"};

countRuns(ee, -2) // n is negative

countRuns(ee, 0) // n is 0, nothing

countRuns(ee, 6) // normal case. Three different elements

string ff[6] = { "zero", "one", "two", "three", "four", "five"};

flip(ff, -2) // n is negative

flip(ff, 0) // n is zero

flip(ff, 6) // normal case

string gg[6] = { "zero", "one", "two", "three", "four", "five"};

string hh[6] = { "zero", "one", "two", "wired", "four", "five"};

differ(gg, -2, hh, 6) // when one of n is negative

differ(gg, 6, hh, -2) // when one of n is negative

differ(gg, 0, hh, 0) // no difference

differ(gg, 6, hh, 6) // normal case

differ(gg, 6, hh, 2) // normal case, when one string is shorter than the same part.

string ii[6] = { "zero", "one", "two", "three", "four", "five"};

string jj[3] = { "two", "three", "four"};

string kk[5] = { "two", "four", "two", "three", "four" };

subsequence(ii, -2, jj, 3) // when one of n is negative

subsequence(ii, 6, jj, -2) // when one of n is negative

subsequence(ii, 0, jj, 3) // when ii has length 0, no comparison

subsequence(ii, 6, jj, 0) // jj at 0 can be any string

subsequence(ii, 6, jj, 3) // normal case

subsequence(kk, 5, jj, 3) // two times of match where the first match is not valid

string mm[6] = { "beta", "one", "three", "six", "four", "five"};

string nn[6] = { "zero", "one", "three", "six", "four", "five"};

lookupAny(ii, -2, mm, 6) // when one of n is negative

lookupAny(ii, 6, mm, -2) // when one of n is negative

lookupAny(ii, 6, mm, 0) // zero length, no comparison

lookupAny(ii, 0, mm, 6) // zero length, no comparison

lookupAny(ii, 6, mm, 6) // normal case 1

lookupAny(ii, 6, nn, 6) // normal case 2

separate(ii, -2, "rom") // when n is negative

separate(ii, 0, "rom") // n if there are no such elements

separate(ff, 6, "rom") // normal case 1

separate(ii, 6, "four") // normal case 2, when separator is one of element

I wrote a function to check the test cases.

int main()

{

string a[6] = { "zero", "one", "two", "three", "four", "five"};

assert(appendToAll(a, -2, "???") == -1); //n is negative, return -1

assert(appendToAll(a, 0, "???") == 0); // n is 0, return 0

assert(appendToAll(a, 6, "???") == 6 && a[0] == "zero???" && a[1] == "one???" && a[2] == "two???" && a[3] == "three???" && a[4] == "four???" && a[5] == "five???"); //normal case

string bb[6] = { "zero", "one", "two", "three", "four", "five"};

assert(lookup(bb, -2, "one") == -1); // n is negative, return -1

assert(lookup(bb, 2, "zero") == 0); // normal case

assert(lookup(bb, 6, "seven") == -1); // normal case: no match

string cc[6] = { "zero", "one", "two", "three", "four", "five"};

assert(positionOfMax(cc, 0) == -1); // n is negative or 0, there is no max, return -1

assert(positionOfMax(cc, 1) == 0); // n is 1, there is no max, return the first one

assert(positionOfMax(cc, 6) == 0); // normal case, zero is the max

string dd[6] = { "zero", "one", "two", "three", "four", "five"};

assert(rotateLeft(dd, -2, 0) == -1); // n is negative

assert(rotateLeft(dd, 6, 2) == 2 && dd[0] == "zero" && dd[1] == "one" && dd[2] == "three" && dd[3] == "four" && dd[4] == "five" && dd[5] == "two"); // normal case

string ee[6] = { "zero", "one", "one", "two", "two", "two"};

assert(countRuns(ee, -2) == -1); // n is negative

assert(countRuns(ee, 0) == 0); // n is 0, nothing

assert(countRuns(ee, 6) == 3); // normal case. Three different elements

string ff[6] = { "zero", "one", "two", "three", "four", "five"};

assert(flip(ff, -2) == -1); // n is negative

assert(flip(ff, 0) == 0); // n is zero

assert(flip(ff, 6) == 6 && ff[0] == "five" && ff[1] == "four" && ff[2] == "three" && ff[3] == "two" && ff[4] == "one" && ff[5] == "zero"); // normal case

string gg[6] = { "zero", "one", "two", "three", "four", "five"};

string hh[6] = { "zero", "one", "two", "wired", "four", "five"};

assert(differ(gg, -2, hh, 6) == -1); // when one of n is negative

assert(differ(gg, 6, hh, -2) == -1); // when one of n is negative

assert(differ(gg, 0, hh, 0) == 0); // no difference

assert(differ(gg, 6, hh, 6) == 3); // normal case

assert(differ(gg, 6, hh, 2) == 2); // normal case, when one string is shorter than the same part.

string ii[6] = { "zero", "one", "two", "three", "four", "five"};

string jj[3] = { "two", "three", "four"};

string kk[5] = { "two", "four", "two", "three", "four" };

assert(subsequence(ii, -2, jj, 3) == -1); // when one of n is negative

assert(subsequence(ii, 6, jj, -2) == -1); // when one of n is negative

assert(subsequence(ii, 0, jj, 3) == -1); // when ii has length 0, no comparison

assert(subsequence(ii, 6, jj, 0) == 0); // jj at 0 can be any string

assert(subsequence(ii, 6, jj, 3) == 2); // normal case

assert(subsequence(kk, 5, jj, 3) == 2); // two times of match where the first match is not valid

string mm[6] = { "beta", "one", "three", "six", "four", "five"};

string nn[6] = { "zero", "one", "three", "six", "four", "five"};

assert(lookupAny(ii, -2, mm, 6) == -1); // when one of n is negative

assert(lookupAny(ii, 6, mm, -2) == -1); // when one of n is negative

assert(lookupAny(ii, 6, mm, 0) == -1); // zero length, no comparison

assert(lookupAny(ii, 0, mm, 6) == -1); // zero length, no comparison

assert(lookupAny(ii, 6, mm, 6) == 1); // normal case 1

assert(lookupAny(ii, 6, nn, 6) == 0); // normal case 2

assert(separate(ii, -2, "rom") == -1);// when n is negative

assert(separate(ii, 0, "rom") == 0); // n if there are no such elements

assert(separate(ff, 6, "rom") == 3 && ff[0] == "five" && ff[1] == "four" && ff[2] == "one" && ff[3] == "three" && ff[4] == "two" && ff[5] == "zero");// normal case 1

assert(separate(ii, 6, "four") == 1 && ii[0] == "five" && ii[1] == "four" && ii[2] == "one" && ii[3] == "three" && ii[4] == "two" && ii[5] == "zero");// normal case 2, when seperator is one of element

cout << "All tests succeeded" << endl;

}