RotateLeft()

string g[4] = { "greg", "gavin", "fiona", "kevin" };

assert(rotateLeft(g, 4, 1) == 1 && g[1] == "fiona?" && g[3] == "gavin?");

The following conditions are:

when size of the size of string (interesting elements) is less than 0

assert(rotateLeft(g,-5,4)==-1);

When input of pos is less than 0

assert(rotateLeft(g,4,-1)==-1);

When size of interesting element is 0

return 0 when there’s only one element in the array

Return -1 when size = 0, because there exist no string to rotate left. Wrong Input

assert(rotateLeft(g,0,0)==-1);

When pos is greater than size

assert(rotateLeft(g,3,4)==-1&&rotateLeft(g,3,3)==-1);

Differ()

Test whether the

string h[7] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona" };

string g[4] = { "greg", "gavin", "fiona", "kevin" };

assert(differ(h, 4, g, 4) == 2);

When there exist overlapping elements which are not in the right order

string m[3]={"greg", "fiona", "kevin"};

assert(differ(h, 4, m, 3) == 1);

When input size is less than 0

assert(differ(h, 4, m, -3) == -1);

When input size is 0

 assert(differ(h, 4, m, 0) == 0);

appendToAll()

 string g[4] = { "greg", "gavin", "fiona", "kevin" };

The following conditions are:

assert(appendToAll(g, 4, "?") == 4 && g[0] == "greg?" && g[3] == "kevin?");

When size is equal to or lower than 0

assert(appendToAll(g, 0,"!")==0&&appendToAll(g, -1,"!")==-1);

lookup( )

string h[7] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona" };

When item is found:

    assert(lookup(h, 7, "eleni") == 5);

assert(lookup(h, 7, "ed") == 2);

When item is not found:

assert(lookup(h, 2, "ed") == -1);

When item not found(target is an empty string)

assert(lookup(h, 2, "") == -1);

countRuns( )

Normal Condition:

string d[5] = { "gavin", "gavin", "gavin", "xavier", "xavier" };

    assert(countRuns(d, 5) == 2);

When all elements are same

string k[4] = { "gavin", "gavin", "gavin", "gavin"};

    assert(countRuns(k, 4) == 1);

When all elements are different

(PLEASE PROFESSOR SMALLBERG DON'T KILL ME)

string berg[5] = {"Smallberg","Smallerberg","Mediumberg","Hugeberg","Giantberg"};

assert(countRuns(berg, 5) == 5);

When number of elements less than or equal to 0

assert(countRuns(berg, 0) == 0);

assert(countRuns(berg, -3) == -1);

flip( )

When number of elements is

1.positive

2.less than or equal to 0

string f[3] = { "fiona", "ed", "john" };

    assert(lookupAny(h, 7, f, 3) == 2);

    assert(flip(f, 3) == 3 && f[0] == "john" && f[2] == "fiona");

assert(flip(f, -3) == -1);

PS: To make life easier, I designed an extra function to exchange the position of two string in an array, which is used in flip()

swapString(string& a, string& b);

int subsequence( )

string str1[7]={"ab","cd","ef","ab","cd","ef","gh"};

string str2[4]={"ab","cd","ef","gh"};

When one is the subsequence of the other

assert(subsequence(str1,7,str2,4)==3);

When it is not

assert(subsequence(str1, 7, str2, 3)==0);

Input size is 0

assert(subsequence(str1,0,str2,0)==0 && subsequence(str1,-4,str2,0)==-1));

lookupAny()

string h[7] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona" };

string f[3] = { "fiona", "ed", "john" };

When there exist a coincident element

assert(lookupAny(h, 7, f, 3) == 2);

When there doesn’t exist a common element, or interesting element is smaller than zero

assert(lookupAny(h, 1, f, 3) == -1 && lookupAny(h, 7, f, -3)==-1);

positionOfMax()

string h[7] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona" };

Normal condition:

assert(positionOfMax(h, 7) == 3);

When interesting element is smaller than zero

assert(positionOfMax(h, -2) == -1);

divide()

The following conditions are:

1. When some elements are equal to the divider while others are not
2. When all the elements equal the divider
3. When none of the elements equals the divider
4. When number of interesting elements is less than 0

string str3[14]={"n","d","f","s","p","i","o","s","ge","f","a","o","o","c"};

    string str4[10]={"n","n","n","n","n","n","n"};

    string str5[5]={"c","c","c","c","c"};

    divide(str3,14,"o");

    divide(str4,7,"n");

    divide(str5,5,"n");

    int i = 0;

    while(i<14){

        cerr<<str3[i]<<" ";

        i++;

    }//str3 should equal {"n","d","f","c","ge","i","a","f","o","o","o","s","p","s"};

    cerr<<endl;

    i=0;

    while(i<7){

        cerr<<str4[i]<<" ";

        i++;

    }//str4 should equal {"n","n","n","n","n","n","n"};

    cerr<<endl;

    i=0;

    while(i<5){

        cerr<<str5[i]<<" ";

        i++;

    }//str5 should equal {"c","c","c","c","c"};

    cerr<<endl;

    cerr<<endl;

    cerr << "All tests succeeded" << endl;

}

Attached is the full test case:

int main()

{

    string h[7] = { "greg", "gavin", "ed", "xavier", "", "eleni", "fiona" };

    assert(lookup(h, 7, "eleni") == 5);

    assert(lookup(h, 7, "ed") == 2);

    assert(lookup(h, 2, "ed") == -1);

    assert(lookup(h, 2, "") == -1);

    assert(positionOfMax(h, 7) == 3 && positionOfMax(h, -2) == -1);

    string g[4] = { "greg", "gavin", "fiona", "kevin" };

    assert(differ(h, 4, g, 4) == 2);

    assert(appendToAll(g, 4, "?") == 4 && g[0] == "greg?" && g[3] == "kevin?");

    assert(appendToAll(g, 0,"!")==0&&appendToAll(g, -1,"!")==-1);

    assert(rotateLeft(g, 4, 1) == 1 && g[1] == "fiona?" && g[3] == "gavin?");

    assert(rotateLeft(g,-5,4)==-1 && rotateLeft(g,4,-1)==-1 && rotateLeft(g,0,0)==-1);

    assert(rotateLeft(g,3,4)==-1&&rotateLeft(g,3,3)==-1);

    string m[3]={"greg", "fiona", "kevin"};

    assert(differ(h, 4, m, 3) == 1);

    assert(differ(h, 4, m, -3) == -1);

    assert(differ(h, 4, m, 0) == 0);

    string e[4] = { "ed", "xavier", "", "eleni" };

    assert(subsequence(h, 7, e, 4) == 2);

    string d[5] = { "gavin", "gavin", "gavin", "xavier", "xavier" };

    assert(countRuns(d, 5) == 2);

    string k[4] = { "gavin", "gavin", "gavin", "gavin"};

    assert(countRuns(k, 4) == 1);

    string f[3] = { "fiona", "ed", "john" };

    assert(lookupAny(h, 7, f, 3) == 2);

    assert(lookupAny(h, 1, f, 3) == -1 && lookupAny(h, 7, f, -3)==-1);

    assert(flip(f, 3) == 3 && f[0] == "john" && f[2] == "fiona");

    assert(flip(f, -3) == -1);

    assert(divide(h, 7, "fiona") == 3);

    //Additional Tests

    string str1[7]={"ab","cd","ef","ab","cd","ef","gh"};

    string str2[4]={"ab","cd","ef","gh"};

    assert(subsequence(str1,7,str2,4)==3 && subsequence(str1, 7, str2, 3)==0 && subsequence(str1,0,str2,0)==0 && subsequence(str1,-4,str2,0)==-1);

    //test the string modified by divide()

    string str3[14]={"n","d","f","s","p","i","o","s","ge","f","a","o","o","c"};

    string str4[10]={"n","n","n","n","n","n","n"};

    string str5[5]={"c","c","c","c","c"};

    divide(str3,14,"o");

    divide(str4,7,"n");

    divide(str5,5,"n");

    int i = 0;

    while(i<14){

        cerr<<str3[i]<<" ";

        i++;

    }//str3 should equal {"n","d","f","c","ge","i","a","f","o","o","o","s","p","s"};

    cerr<<endl;

    i=0;

    while(i<7){

        cerr<<str4[i]<<" ";

        i++;

    }//str4 should equal {"n","n","n","n","n","n","n"};

    cerr<<endl;

    i=0;

    while(i<5){

        cerr<<str5[i]<<" ";

        i++;

    }//str5 should equal {"c","c","c","c","c"};

    cerr<<endl;

    cerr<<endl;

    cerr << "All tests succeeded" << endl;

}