Project 4

Most of my trouble can be pinpointed due to my misuse of loops. My first problem with this was with the lastOccurrence function. It returned the index of the target just fine, but if the target was not found, it did not return -1. This is because I used an else statement in attempt to return -1, which made the loop only run once. I fixed this bug by taking out the else statement, and instead, outside of the loop I returned -1. That way, if the loop broke, in other words, if the target was not found, then the function would return -1.

I encountered nearly the same exact bug with the void function hasNoDuplicates. This involved a logic error as well, as I mixed up when false should be returned and true should be returned. When the loop breaks, the function returns true, meaning that there are no duplicates.

Other obstacles were mainly meticulous errors that I found after pouring over the spec. For example, for int shiftRight, I questioned if both n is 0 and amount is 0, what should be returned? After asking Howard, he said it should returned -1. But, I already coded that if amount and n are equal, it should return 0. This was easily fixed by refining the if statement that indicates that if amount and n are equal, by adding the limitation that n and amount cannot be 0.

Overall, the functions were relatively easy to code, with the exception of unionWithNoDuplicates. This function was most difficult for me by far. However, after drawing a diagram on paper, slowly but surely, I was able to translate my logic into my code. Originally, I misunderstood the spec, and programmed the function to check if there were duplicates in array1, and then array2. If there were duplicates within each array, it would not be appended into the resultingString at all. After reading the spec again, I realized that this is not what was asked of me. Rather, it should take the unique elements of each array and combine them into one array, resultingString. Once I realized this, I changed my approach by dumping array1 into resultingString, and by incrementing the position of array1 and resultingString, checking if the element from array1 was already in resultingString, then it should continue searching the string. If the element was not present, then it should copy then element into resultingString. Then, I did the same with another loop in the function for array2.

Test Cases

Function 1

**int locateMinimum( const string** **array[ ],int n );**

void testlocateMinimum()

{

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

    assert(locateMinimum(alphabet, 0)==-1);//test if n is 0, return -1.

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

    string capitalAlpha[3] = {"A", "AA", "B"};

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

    assert(locateMinimum(capitalAlpha, 2)==0);

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

    assert(locateMinimum(outOfOrderAlphabet, 0)==-1);

    assert(locateMinimum(outOfOrderAlphabet, 3)==1);

    string aAb[3] = {"a", "b", "A"};

    assert(locateMinimum(aAb, 3)==2);

    assert(locateMinimum(aAb, 1)==0);

    string crazyAlpha[7]={"B", "AA", "ccC", "bbbB", "A", "ccCzzzzz", "ccCabc"};

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

    assert(locateMinimum(crazyAlpha, 3)==1);

    string otherAlpha[4] = {"a", "a", "b", "a"};    //test if that there are duplicates of smallest item, return minimum index

    assert(locateMinimum(otherAlpha, 4)==0);

    assert(locateMinimum(otherAlpha, -5)==-1);  //test if n is less than 0, return -1

    string family[7] = {"val", "stef", "baba", "abe" , "vince", "emilio", "christian"};

    assert(locateMinimum(family, 7)==3);

    assert(locateMinimum(family, 2)==1);        //test to see if works with limitation of n.

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

    assert(locateMinimum(family, 4)==3);

    string people[5] = { "samwell", "jon", "margaery", "daenerys", "tyrion" };

    assert(locateMinimum(people, 0)==-1);

    assert(locateMinimum(people, 5)==3);

    assert(locateMinimum(people, 3)==1);

    cerr << "locateMinimum passes!" << endl;

}

Function 2

**int findLastOccurrence( const string array[ ], int n, string target );**

void testfindLastOccurrence()

{

    string data[ 4 ] = { "howard", "ucla", "howard", "ucla" };

    assert(findLastOccurrence(data, 4, "howard")==2);

    string largeAlpha[3] = {"A", "A", "A"};

    assert(findLastOccurrence(largeAlpha, 4, "A")==2);  //return largest index that holds target in the array

    assert(findLastOccurrence(largeAlpha, 4, "B")==-1); //if target not found, return -1.

    string alphabet[5] = {"a", "a", "a", "a", "a"};

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

    assert(findLastOccurrence(alphabet, 5, "b")==-1);   //if target not found, return -1.

    cerr<< "second function passes!" <<endl;

}

Function 3

**int flipAround( string** **array[ ],int** **n );**

void testflipAround()

{

    string nothing[1] = {""};           //test if nothing returns 0

    assert(flipAround(nothing, 1)==0);

    string one[1] = {"one"};            //if one element, return 0

    assert(flipAround(one, 1)==0);

    string negative[1] = {"negative"};  //if n is negative, return 0

    assert(flipAround(negative, -1)==0);

    string alphabet[2] = {"a", "b"};

    assert(flipAround(alphabet, 2) == 1);       //return amount of flips:

    string alphabet2[9] = {"a", "b", "c", "d", "e", "f", "g", "h", "i"};

    assert(flipAround(alphabet2, 9) == 4);

    assert(flipAround(alphabet2, 3) == 1);

    assert(flipAround(alphabet2, 4) == 2);

    assert(flipAround(alphabet2, 6) == 3);

    string alphabet3[12] = {"a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l"};

    assert(flipAround(alphabet3, 9) == 4);

    string family[7] = {"val", "stef", "baba", "abe" , "vince", "emilio", "christian"};

    assert(flipAround(family, 7)==3);

    string people[ 5 ] = { "samwell", "jon", "margaery", "daenerys", "tyrion" } ;

    assert(flipAround(people, 5)==2);

    cerr << "third function passes!"<< endl;

}

Function 4  
**bool hasNoDuplicates( const string array[ ], int  n );**

void testhasNoDuplicates()

{

    string people[ 5 ] = { "samwell", "jon", "margaery", "daenerys", "tyrion" } ;

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

    string alphabet[12] = {"a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l"};

    assert(hasNoDuplicates(alphabet, -1) == false);     //if n is negative, return false.

    assert(hasNoDuplicates(alphabet, 0) == true);       //if n is 0, return true

    assert(hasNoDuplicates(alphabet, 1) == true);       //if n is 1, return true

    assert(hasNoDuplicates(alphabet, 12) == true);      //no duplicates, return true.

    string repeatAlpha[5] = { "a", "b", "c", "c", "c"};

    assert(hasNoDuplicates(repeatAlpha, 3) == true);    //when n is 3, no duplicates.

return true

    assert(hasNoDuplicates(repeatAlpha, 5) == false);   //when n is 5, there are

duplicates. return false.

    string repeatAlpha2[5] = { "a", "a", "a", "a", "a"};

    assert(hasNoDuplicates(repeatAlpha2, 3) == false);

    string repeatAlpha3[5] = { "a", "b", "c", "b", "d"};

    assert(hasNoDuplicates(repeatAlpha3, 5) == false);  //test to see if returns false if duplicates are not adjacent.

    string family[7] = {"val", "stef", "baba", "abe" , "vince", "emilio", "christian"};

    assert(hasNoDuplicates(family, 7) == true);

    cerr<< "fourth function passes!" <<endl;

}

Function 5

**void unionWithNoDuplicates( const string array1[ ], int n1, const string array2[ ], int n2, string resultingString[ ], int& resultingSize );**

    string data[ 3 ] = { "data", "data",  "howard"};

    string people[ 2 ] = { "samwell", "samwell"};

    string resultingString[4];

    int size = 5;

    unionWithNoDuplicates(data, 3, people, 2, resultingString, size);

   cout<<size<< endl; //see if correct size is printed. It does.

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

     {

     cout<<resultingString[i]<<endl; //see if correct array is printed. It does.

     }

    string alpha[ 4 ] = { "a", "b", "a"};

    string alpha2[ 2 ] = { "c", "d"};

    string resultingString[4];

    int size = 5;

    unionWithNoDuplicates(alpha, 3, alpha2, 2, resultingString, size);

   cout<<size<< endl; //see if correct size is printed. It does.

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

     {

     cout<<resultingString[i]<<endl; //see if correct array is printed. It does.

     }

    string number[ 5 ] = { "1", "2", "3", "4", "5"};

    string numberTwo[ 2 ] = { "6", "7"};

    string resultingString[4];

    int size = 5;

    unionWithNoDuplicates(number, 5, numberTwo, 2, resultingString, size);

   cout<<size<< endl; //see if correct size is printed. It does.

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

     {

     cout<<resultingString[i]<<endl;

     }

    string number[ 5 ] = { "1", "2", "3", "4", "5"};

    string numberTwo[ 2 ] = { "6", "7"};

    string resultingString[4];

    int size = 5;

    unionWithNoDuplicates(number, 5, numberTwo, 2, resultingString, size);

    cout <<size<< endl; //see if correct size is printed. It does.

    for(int i=0;i<size;i++)

     {

     cout<<resultingString[i]<<endl; //see if correct array is printed. It does.

     }

Function 6

**int shiftRight( string array[ ], int n, int amount, string   placeholderToFillEmpties );**

void testshiftRight()

{

    string people[ 5 ] = { "samwell", "jon", "margaery", "daenerys", "tyrion" } ;

    assert(shiftRight(people, 5, 3, "foo")==2);

    assert(shiftRight(people, 4, 2, "foo")==2);

    assert(shiftRight(people, 5, 5, "foo")==0);

    string alpha[3]={"a", "b", "c"};

    assert(shiftRight(alpha, 3, 2, "foo")==1);

    assert(shiftRight(alpha, -1, 2, "foo")==-1); //if n is less than 0, return -1.

    assert(shiftRight(alpha, 3, 4, "foo")==-1); //if amount is greater than n, return -1.

    assert(shiftRight(alpha, 3, -1, "foo")==-1);

    string family[7] = {"baba", "stef", "val", "abe" , "vince", "emilio", "christian"};

    assert(shiftRight(family, 3, 2, "foo" )==1);

    assert(shiftRight(family, 4, 3, "foo" )==1);

    assert(shiftRight(family, 5, 4, "foo" )==1);

    assert(shiftRight(family, 6, 5, "foo" )==1);

    assert(shiftRight(family, 7, 6, "foo" )==1);

    assert(shiftRight(family, 8, 7, "foo" )==1);    //cannot do anything about this

    assert(shiftRight(family, 7, 2, "foo" )==5);

    assert(shiftRight(family, 7, 4, "foo" )==3);

    cerr << "shift right passes!"<<endl;

}

Function 7

**bool isInIncreasingOrder( const string array[ ], int  n );**

void testisInIncreasingOrder()

{

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

    assert(isInIncreasingOrder(alphabet, 5)==false);

    string repeatAlpha[5] = { "a", "b", "c", "c", "c"};

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

    assert(isInIncreasingOrder(repeatAlpha, 5)==false);     //equivalence is not increasing order

    string capitalAlpha[3] = {"A", "AA", "B"};

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

    string otherAlpha[5] = {"C", "CC", "D", "A", "B"};  //making sure that it can read just first 3 of array

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

    assert(isInIncreasingOrder(otherAlpha, -1)==false); //if n is less than 0, return false

    assert(isInIncreasingOrder(otherAlpha, 1)==true);   //if n is 1, return true

    assert(isInIncreasingOrder(otherAlpha, 0)==true); //if n is 0, return true

    string things[3] = {"apple", "ball", "ball"};

    assert(isInIncreasingOrder(things, 2)==true);

    assert(isInIncreasingOrder(things, 3)==false);

    string people[5] = { "samwell", "jon", "margaery", "daenerys", "tyrion"};

    assert(isInIncreasingOrder(people, 5)==false);

    string family[7] = {"baba", "stef", "val", "abe" , "vince", "emilio", "christian"};

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

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

    cerr << "last function passses!" <<endl;

}