report.docx

1. Notable Obstacles
   1. The first major obstacle that I encountered occurred when I was reorganizing the elements of a string. I did not know whether it was more efficient to create a second, temporary, array or to find a way to use the original array itself. I decided on using temporary arrays to store my reorganized elements.
   2. A second obstacle that I faced arose when trying to figure out the best way to nest my loops for some of the functions. Figuring out the best way to nest the loops was definitely a challenge. I made my final decisions on the most concise and efficient methods of reaching the proper returns. For the large majority of the functions, I used for loops, with the exception of 2 or 3 while loops.
2. Test Cases
   1. int appendToAll(string a[], int n, string value);

string people[5] = { "clark", "peter", "diana", "tony", "selina" };

int j = appendToAll(people, 5, "!!!"); //basic output

int j = appendToAll(people, 8, "!!!"); //out of bounds

int j = appendToAll(people, -3, "!!!"); //out of bounds

* 1. int lookup(const string a[], int n, string target);

string h[7] = { "selina", "reed", "diana", "tony", "", "logan", "peter" };

assert(lookup(h, 7, "logan") == 5); //basic output

assert(lookup(h, 2, "diana") == -1); //target not found

assert(lookup(h, -3, "diana") == -1); //out of bounds

assert(lookup(h, 7, "LOgan") == -1); //misspelled, not found

* 1. int positionOfMax(const string a[], int n);

string hero[6] = { "clark", "peter", "reed", "tony", "diana", "bruce" };

int k = positionOfMax(hero, 6); //basic output

int k = positionOfMax(hero, -3); //out of bounds

* 1. int rotateLeft(string a[], int n, int pos);

string super[5] = { "logan", "clark", "peter", "sue", "reed" };

int m = rotateLeft(super, 5, 1); // basic output

int m = rotateLeft(super, -5, 1); // n cannot be less than zero

int m = rotateLeft(super, 5, -3); // pos cannot be less than zero

int m = rotateLeft(super, 5, 8); // pos must be less than n

* 1. int countRuns(const string a[], int n);

string d[9] = { "tony", "bruce", "steve", "steve", "diana", "diana", "diana", "steve", "steve" };

int p = countRuns(d, 9); //basic output

int p = countRuns(d, -2); //out of bounds

* 1. int flip(string a[], int n);

string folks[6] = { "bruce", "steve", "", "tony", "sue", "clark" };

int q = flip(folks, 4); //basic output

int q = flip(folks, -5); //out of bounds

* 1. int differ(const string a1[], int n1, const string a2[], int n2);

string folks[6] = { "bruce", "steve", "", "tony", "sue", "clark" };

string group[5] = { "bruce", "steve", "clark", "", "tony" };

int s = differ(folks, 6, group, 5); //basic output

int s =differ(folks, -2, group, -1);//n1 and n2 can’t be less than zero

int s =differ(folks, 1, group, 1);//a1 and a2 have the same elements

int s=differ(folks, 1, group, 0);//should return output -1 since no equality of elements

* 1. int subsequence(const string a1[], int n1, const string a2[], int n2);

string names[10] = { "logan", "reed", "sue", "selina", "bruce", "peter" };

string names1[10] = { "reed", "sue", "selina" };

int t = subsequence(names, 6, names1, 3); // basic output

int u = subsequence(names, -3, names1, -2); //n1 and n2 cannot be negative

int u = subsequence(names, 6, names2, 7); //n2 must be less than or equal to n1

int u = subsequence(names, 6, names2, 0); //n2 is 0, return should be 0 as well due to the spec

* 1. int lookupAny(const string a1[], int n1, const string a2[], int n2);

string names[10] = { "logan", "reed", "sue", "selina", "bruce", "peter" };

string set1[10] = { "clark", "bruce", "selina", "reed" };

string set2[10] = { "tony", "diana" };

int v = lookupAny(names, 6, set1, 4); //basic output

int w = lookupAny(names, 6, set2, 2); //has no common elements

int v = lookupAny(names, -3, set1, -4); // n1 and n2 can’t be negative

* 1. int split(string a[], int n, string splitter);

string hero[6] = { "clark", "peter", "reed", "tony", "diana", "bruce" };

int x = split(hero, 6, "logan"); //basic output

int y = split(hero, 0, "logan"); //when n=0

int y = split(hero, -3, "logan"); //n cannot be negative