Zach North

603-885-768

*Project 5 Report*

Obstacles were plentiful in this project. The first function was a complete quagmire for me and I spent maybe 10 hours just writing the code, to say nothing about debugging. The syntax of cstrings was very annoying, especially since I missed a key lecture explaining them (won’t do that again!) I found the hardest aspect to be mapping out the logic in my head – it was confusing and difficult, and hard to picture analytically. Multiple loops simply don’t register very well in my brain. Eventually I found writing it out in pseudocode made it slightly easier and figured it out. The second function was a relative breeze, except for one incident when I used strcat instead of strcpy and spent an hour combing my code for the three-letter error that was causing so many infinite loops. Once I fixed it the rest of the function came naturally, with few problems.

Pseudocode:

*standardizeCriteria*loop through words  
 if the corresponding distance is not greater than 0  
 remove this criteria (distance, word1, and word2) and shift following distances to overlap  
 reduce the total criteria by 1, and reduce the current word by 1 to keep it from jumping ahead;  
 loop through letters  
 if current letter of word1 == ‘\0’  
 stop stepping through letters of word1;  
 if current letter of word2 == ‘\0’  
 stop stepping through letters of word2;  
 if current letter of word1 is uppercase  
 convert it to lowercase;  
 if current letter of word2 is uppercase  
 convert it to lowercase;  
 if current letter of word 1 is not a letter  
 remove this criteria (distance, word1, and word2) and shift following distances to overlap;  
 reduce the total criteria by 1, and reduce the current word by 1 to keep it from jumping ahead;  
 break;  
 if current letter of word2 is not a letter  
 remove this criteria (distance, word1, and word2) and shift following distances to overlap  
 reduce the total criteria by 1, and reduce the current word by 1 to keep it from jumping ahead;  
 break;  
if any 2 criteria have the same word1 and word2 value  
 remove the criteria with a smaller distance value;  
return nCriteria;  
  
*calculateScore*  
declare a two variable array to hold the letters split into separate words in document;  
loop through each word integer for the array  
 declare a blank temp array to hold the current word;  
 loop through characters in document  
 if the current character is ‘\0’ or the current character is 200 (the limit for the size of document)  
 set a Boolean to indicate the end has been reached;  
 break out of the letters loop;  
 if the current character is a space  
 attaches a null char on the end of temp;  
 increments the current character;  
 breaks out of the letters loop;  
 else if the current character is not a letter  
 do nothing;  
 else if the current character is uppercase  
 add the lowercase version of the letter onto temp;  
 increment the position in the temp array to the next character;  
 else (the character is a lowercase letter)  
 add the character onto temp;  
 increment the position in the temp array to the next character;  
 if the first character in temp is ‘\0’  
 decrement the word integer by 1 (skip this character) (this is to prevent empty words in testDoc);  
 else  
 increment the integer numberOfWords by 1;  
 copy the word in temp to the current word element in testDoc;  
 if the reachedEnd Boolean is true  
 break out of the words loop;  
copy the empty string into the word after the final word in the words loop (this is mainly for debugging);  
initialize the score counter to 0;  
loop through each criteria  
 loop through each word in testDoc  
 if word1 in the current criteria == the current word in testDoc  
 initialize a Boolean (checking if word2 at current criteria is matched) to false;  
 loop through the words following the matched word, up to the distance in the distance array  
 if the second word matches a word in testDoc  
 increment the score by 1;  
 set the word2Match Boolean to true;  
 break out of the second word check loop;  
 if word2Match is true  
 break out of the word1 loop (this prevents multiple matches for one criteria);  
return the accumulated score;

Test Cases

To thoroughly test this program one would need to split the test cases into two separate inputs: one testing *standardizeCriteria*, and the other testing *calculateScore*

standardizeCriteria:  
possible input for word1:  
{*test test TEST T3ST test-1 test one abc }*word2:  
*{one one two three abc test abc}*what is being tested:**default duplicate uppercase numbers symbols spaces repeats**

The value of distance is not relevant for this test, and is such initialized so every value is 1.The correct return should be as follows:  
word1:  
{*test test}*word2:  
*{one two}*  
  
A successful run of this case would indicate that standardizeCriteria handles conversion to lowercase and removal of any strings with numbers, symbols, spaces, identical word1/word2 repeats, or any duplicates.

Another test to run is to see if the program handles empty strings or symbols correctly.

Word1:  
{ *(blank) \0 \t \n}*word2:  
{*one two three four}*  
testing**:  
blank nulls tabs newlines**

The correct return should eliminate all of these values.

To fully test this function however, we must make sure it handles problems in both word1 and word2. Thus we would repeat the above two tests, except with the *word1* input and *word2* input flipped. The values returned should be the same, except reversed. If this is the case it would indicate that both strings are being tested and having criteria removed in an identical fashion.

Now it is necessary to test that the function removes the duplicates with *lower distance values*, and also that it correctly handles removal of nonpositive distance values.  
  
For this, we could test using the following:  
word1:  
{*test test test test test test test}*word2:  
*{one two two three three four five}*distance:  
*{2 3 5 -1 4 0 -3}*what is being tested:  
**default 🡪 duplicate negative duplicate zero negative**The function should return:  
word1:  
*{test test test}*word2:  
*{one two three}*distance:  
*{2 5 4}*

This would indicate standardizeCriteria successfully handles removing negative values, zero values, and that any duplicates would have the element with the *smaller corresponding distance value* removed.

calculateScore:

This function is relatively more straightforward to test. All of the word1, word2, and distance array inputs must be syntactically correct, so it basically just involves modifying *document* to test whether the function returns a correct result for score.

A sample input would be alike to the ones in the sample on the course website:  
word1:   
*{ “mad” “deranged” “nefarious” “have”}*word2:  
*{“scientist” “robot” “plot” “mad”}*distance:  
*{2 4 1 13}*The only thing that would vary would be the value of document, and the score returned.  
  
example documents:  
“The mad UCLA scientist unleashed a deranged evil giant robot.” //normal.  
“The mad UCLA scientist unleashed a deranged robot.” //tests that the function handles excess spaces

“\*\*\*\*\*2010\*\*\*\*\*” //tests that the function handles symbols and numbers.  
“ What a NEFARIOUS plot!” //checks that the function handles excess spaces and uppercase letters.  
“deranged deranged robot deranged robot robot” //checks that the function handles duplicates  
“Two mad scientists have deranged-robot fever.” //checks that the function handles symbols again  
  
These documents should return, in order:  
2  
2  
0  
1  
1  
0.

If all of the above tests succeed and return the specified results, it is safe to declare the function debugged and fully functional because every error possibility has been tested extensively.