Oo, Thant Zin (Andy)

CS 31, Section 2F

Project 5 Report

1. I had difficulty implementing both functions. The approach I had for function 1 was unnecessarily complicated. Rather than writing a simple function that removed patterns by finding the index of the faulty patterns and shifting everything after it one to the left, I tried to keep track of the number of valid patterns and store valid patterns into a temporary array which I copied back into the original array. However, this approach was messy- it involved 4x the number of lines as my current code. In addition I ran into undefined behavior when I was calling strcpy with the same destination and source memory addresses. For function 2, I kept running into the error message mentioned in the spec, where I was accessing undefined memory beyond the \0 byte. I couldn’t find the bug because it prevented me from using the debugger, so I scratched my approach and recoded much of the looping in function 2. Previously I was using if statements to check if a given index was a space and if the index afterwards was a char. In hindsight that approach was also unnecessarily complex.
2. My program is designed around the following 5 functions:

**int makeProper()**

check for valid nPatterns

repeatedly:

call makeLower to change patterns to lowercase

call removePattern to remove non-valid patterns, ie. non alphabetical characters, negative separation, empty strings

repeatedly:

check if there are duplicate patterns (string equality)

if duplicate, remove pattern with smaller separation

**void makeLower()**

repeatedly:

check for non-alphabetical characters

change all alphabetical characters to lowercase

append null byte to end of c string

**void removePattern()**

repeatedly:

shift all strings in word1 to the right of the string to be removed to the left

repeat for word2

repeat for separation (except with ints)

**int rate()**

check for valid nPatterns

create temporary arrays to store and manipulate elements in document and word1/2

repeatedly:

remove all non alphabetical and non space characters. Change uppercase characters to lowercase

append null byte

repeatedly:

look for pattern pair denoted by index i.

call setWord function to set temporary word array to current word in document

compare current word with pattern pair

if match, increase rating and continue for next pair

return rating

**void setWord()**

repeatedly:

if current character is a space, move on to next character

repeatedly:

store alphabetical letters in temporary word array

append null byte

The two prescribed functions in the spec, makeProper() and rate() call the three void functions to aid their implementation. removePattern() and makeLower() modify the arrays that are passed to them, by removing patterns and changing the patterns to lowercase. Rate() calls setWord(), which modifies the temporary word array but leaves the pattern arrays untouched. setWord also modifies the iterator in the loops that it is called, so that the next word is found when setWord is called again.

1. List of test data used to test rate.cpp:

**const int TEST1\_NRULES = 4;**

**char test1w1[TEST1\_NRULES][MAX\_WORD\_LENGTH + 1] = {**

**"mad", "deranged", "nefarious", "have"**

**};**

**char test1w2[TEST1\_NRULES][MAX\_WORD\_LENGTH + 1] = {**

**"scientist", "robot", "plot", "mad"**

**};**

**int test1dist[TEST1\_NRULES] = {**

**1, 3, 0, 12**

**};**

**Standard case**

assert(rate("The mad UCLA scientist unleashed a deranged evil giant robot.", test1w1, test1w2, test1dist, TEST1\_NRULES) == 2);

**When nPatterns == 0**

assert(rate("The mad UCLA scientist", test1w1, test1w2, test1dist, 0) == 0);

**When nPatterns < 0**

assert(rate("The mad UCLA scientist", test1w1, test1w2, test1dist, -99) == 0);

**Document where there are consecutive space characters**

assert(rate("The mad UCLA scientist unleashed a deranged robot.", test1w1, test1w2, test1dist, TEST1\_NRULES) == 2);

**Non-alphabetical characters**

assert(rate("\*\*\*\* 2018 \*\*\*\*", test1w1, test1w2, test1dist, TEST1\_NRULES) == 0);

**Uppercase pattern match**

assert(rate(" That plot: NEFARIOUS!", test1w1, test1w2, test1dist, TEST1\_NRULES) == 1);

**Multiple pattern occurrence**

assert(rate("deranged deranged robot deranged robot robot", test1w1, test1w2, test1dist, TEST1\_NRULES) == 1);

**Hyphen separation between pattern match. Should not be a match**

assert(rate("That scientist said two mad scientists suffer from deranged-robot fever.", test1w1, test1w2, test1dist, TEST1\_NRULES) == 0);

**char test1w1[5][MAX\_WORD\_LENGTH + 1] = {**

**"mad", "deranged", "nefarious", "have", “plot”**

**};**

**char test1w2[5][MAX\_WORD\_LENGTH + 1] = {**

**"scientist", "robot", "plot", "mad", “nefarious”**

**};**

**int test1dist[5] = {**

**1, 3, 0, 12, 1**

**};**

**Repeated pattern in word1, word2**

assert(makeProper(test1w1, test1w2, test1dist, 5) == 4);

**int test1dist[5] = {**

**1, 3, 0, -12, -1**

**};**

**Negative separation**

assert(makeProper(test1w1, test1w2, test1dist, 5) == 3);

**char test1w2[5][MAX\_WORD\_LENGTH + 1] = {**

**"scientist", "robot", "123plot", "mad", “nefarious”**

**};**

**Make proper handling non-alphabetical characters correctly**

assert(makeProper(test1w1, test1w2, test1dist, 5) == 3 && !strcmp(test1w2[2], “plot”) )