Raj Kumar

Professor Smallberg

Computer Science 31

23 November 2020

Project 5 Report

**Obstacles:**

The largest obstacle was understanding why g31 was giving me an underflow error. I eventually realized that one of my variables was becoming negative, and a negative index of a cstring was being accessed. Visual Studio didn’t seem to care about this, and probably just accessed the memory before the cstring, but g31 firmly said no and gave me an error. I found out that only some compilers that were extremely careful would cause problems with this, probably because some compilers prioritize speed and some stability and debugging power.

My second obstacle was avoiding fencepost type errors when dividing the document cstring into an array of cstrings that held individual words. I kept accidentally going over or under by a character and would store “ord” instead of “word” or “wor” instead of “word”. I also sometimes had uninitialized characters in my cstrings (at the end of the document)

My third obstacle was knowing where to add the zero byte or at times even remembering to add it to a cstring if I used certain functions that I created for the project.

**Pseudocode:**

For cleanupRules, here is my pseudocode:

Loop

Check if a win word is empty string

Remove rule

Check if there are only letters in rule

True: Make string lowercase

False: Remove Rule

Check if wordin and wordout are same in rule

Remove Rule

Loop (to remove duplicates)

Check if wordin between two different rules match

Check if first rule is a one word rule

Remove second rule

Check if second rule is a one word rule

Remove first rule

Check if first and second rule wordouts are same

Remove second rule

Return nRules – numberofRulesRemoved

For determineScore, here is my pseudocode:

Loop to copy document cstring to doc cstring and in the process do:

Remove all non spaces and non-alphabets from the doc

Remove all duplicate spaces

Add zero byte at the end

Loop (to store all words in an array of cstrings called words)

Find next two spaces

Store text between two spaces in array of cstrings called words

Loop

Check if each Rule’s wordin matches with a word in cstring called words

Check if that rule’s wordout is in cstring called words

False: add one to the score

It is a lot less confusing to just read the code and assert statements than to explain the inputs in this case, so I have explained why I used a test case and then showed the code for it. There were also some cases provided by Professor Smallberg included.

**Test Data: (**All test cases were handled in a way the spec requires)

**The green comments give insight on why I did the test**

// cleanupRules Tests

//test 0, to just see correctness

cerr << "test 0" << endl;

char test0win[12][MAX\_WORD\_LENGTH + 1] = {

"confusion", "FAMILY","charm", "hearty","house"};

char test0wout[12][MAX\_WORD\_LENGTH + 1] = {

"", "TIES", "confusion", "hearty","intrigue"};

assert(cleanupRules(test0win, test0wout, 5) == 4);

//test 001, see what happens if I give an empty string pair as input

cerr << "test 001" << endl;

char test001win[2][MAX\_WORD\_LENGTH + 1] = {

"", "FAMILY"

};

char test001wout[2][MAX\_WORD\_LENGTH + 1] = {

"", "TIES",

};

assert(cleanupRules(test001win, test001wout, 2) == 1);

//test 0001, see what happens if I take away a single index at the end from test 01

cerr << "test 0001" << endl;

char test0001win[11][MAX\_WORD\_LENGTH + 1] = {

"", "FAMILY","charm", "hearty","house", "worn-out","family", "charm", "ties", "", "charm",

};

char test0001wout[11][MAX\_WORD\_LENGTH + 1] = {

"", "TIES", "confusion", "hearty","intrigue", "younger","first", "", "family", "frightened", "",

};

assert(cleanupRules(test0001win, test0001wout, 11) == 5);

//test 01 general correctness test

cerr << "test 01" << endl;

char test01win[12][MAX\_WORD\_LENGTH + 1] = {

"", "FAMILY","charm", "hearty","house", "worn-out","family", "charm", "ties", "", "charm","FaMiLy"

};

char test01wout[12][MAX\_WORD\_LENGTH + 1] = {

"", "TIES", "confusion", "hearty","intrigue", "younger","first", "", "family", "frightened", "", "tIeS"

};

assert(cleanupRules(test01win, test01wout, 12) == 5);

//test 02 another general correctness test

cerr << "test 02" << endl;

char test02win[12][MAX\_WORD\_LENGTH + 1] = {

"confusion", "FAMILY","charm", "hearty","house", "worn-out","family", "charm", "ties", "", "charm","FaMiLy"

};

char test02wout[12][MAX\_WORD\_LENGTH + 1] = {

"", "TIES", "confusion", "hearty","intrigue", "younger","first", "", "family", "frightened", "", "tIeS"

};

assert(cleanupRules(test02win, test02wout, 12) == 6);

//test 03 specifically testing to see what happens if there are duplicates of confusion

cerr << "test 03" << endl;

char test03win[12][MAX\_WORD\_LENGTH + 1] = {

"confusion", "CoNFUSION","charm", "hearty","confuSIOn ", "confusion","family", "charm", "ties",

};

char test03wout[12][MAX\_WORD\_LENGTH + 1] = {

"hello", "TIES", "confusion", "hearty","hello", "younger","first", "", "family",

};

assert(cleanupRules(test03win, test03wout, 9) == 6);

//test 04, another duplicates test

cerr << "test 04" << endl;

char test04win[12][MAX\_WORD\_LENGTH + 1] = {

"confusion", "CoNFUSION","charm", "hearty","confuSIOn ", "confusion","family", "charm", "ties", "confusion"

};

char test04wout[12][MAX\_WORD\_LENGTH + 1] = {

"hello", "TIES", "confusion", "hearty","hello", "younger","first", "", "family", ""

};

assert(cleanupRules(test04win, test04wout, 10) == 4);

//test 05, to make sure giving only one empty string returns a 0

cerr << "test 05" << endl;

char test05win[12][MAX\_WORD\_LENGTH + 1] = { "" };

char test05wout[12][MAX\_WORD\_LENGTH + 1] = { "" };

assert(cleanupRules(test05win, test05wout, 1) == 0);

//test 06, to see if there is an error if there is negative input for nRules

cerr << "test 06" << endl;

char test06win[12][MAX\_WORD\_LENGTH + 1] = { "" };

char test06wout[12][MAX\_WORD\_LENGTH + 1] = { "" };

assert(cleanupRules(test06win, test06wout, -1) == 0);

//test 07, another test to see if there is an error if there is negative input and //also 0 input for nRules

cerr << "test 07" << endl;

assert(cleanupRules(test04win, test04wout, -1) == 0);

assert(cleanupRules(test04win, test04wout, 0) == 0);

//Many are tests given by Prof Smallberg for the second function (with exception of my own created tests below)

const int TEST1\_NRULES = 3;

char test1win[TEST1\_NRULES][MAX\_WORD\_LENGTH + 1] = {

"family", "unhappy", "horse",

};

char test1wout[TEST1\_NRULES][MAX\_WORD\_LENGTH + 1] = {

"", "horse", "",};

//another correctness test

assert(cleanupRules(test1win, test1wout, 3) == 3);

//another correctness test

assert(determineScore("horse:stable ratio is 10:1",

test1win, test1wout, TEST1\_NRULES) == 0);

//my own addition to see what happens if I add spaces at the end and a period, I //actually found an important mistake I made with this and the next test

assert(determineScore(" Happy families are all alike ; every unhappy family is unhappy in its own way . ",

test1win, test1wout, TEST1\_NRULES) == 2);

//my own addition to see if I add even more spaces at the end and a period

assert(determineScore(" Happy families are all alike ; every unhappy family is unhappy in its own way . . . ",

test1win, test1wout, TEST1\_NRULES) == 2);

//another correctness test

assert(determineScore("Happy families are all alike; every unhappy family is unhappy in its own way.",

test1win, test1wout, TEST1\_NRULES) == 2);

//My own modified tests are below

//another correctness test to see if it still works if I make nRules one less

assert(determineScore("Happy horses are all alike; every unhappy horse is unhappy in its own way.",

test1win, test1wout, TEST1\_NRULES - 1) == 0);

assert(determineScore("Happy horses are all alike; every unhappy horse is unhappy in its own way.",

test1win, test1wout, TEST1\_NRULES) == 1);

//another correctness test

assert(determineScore("A horse! A horse! My kingdom for a horse!",

test1win, test1wout, TEST1\_NRULES) == 1);

//another correctness test

assert(determineScore("\*\*\*\* 2020 \*\*\*\*",

test1win, test1wout, TEST1\_NRULES) == 0);

//to see if there are any problems if an empty document is entered, as this should //not cause an error and just return 0

assert(determineScore("",

test1win, test1wout, TEST1\_NRULES) == 0);

//to see if having a 251 character document would break anything or cause any out of //bounds calls

assert(determineScore("In Yuval Noah Harari's Big data, google, and the end of free will, Harari describes the concept of Dataism, which he defines as the belief thateverything can be broken down to data at a fundamental level, and that people are algorithms that process ", test1win, test1wout, TEST1\_NRULES) == 0 );

//again make sure that negative nRules doesn’t cause serious problems or errors with very long documents entered

assert(determineScore("In Yuval Noah Harari's Big data, google, and the end of free will, Harari describes the concept of Dataism, which he defines as the belief thateverything can be broken down to data at a fundamental level, and that people are algorithms that process ", test1win, test1wout, -1) == 0);

const int TEST2\_NRULES = 6;

const int TEST3\_NRULES = 5;

char test2win[TEST2\_NRULES][MAX\_WORD\_LENGTH + 1] = {

"family", "unhappy", "horse", "unhappy", "unhappy", "unhappy"

};

char test2wout[TEST2\_NRULES][MAX\_WORD\_LENGTH + 1] = {

"", "horse", "", "happy", "all", "hello"};

char test3win[TEST3\_NRULES][MAX\_WORD\_LENGTH + 1] = {

"family", "unhappy", "horse", "data", "kungpao"

};

char test3wout[TEST3\_NRULES][MAX\_WORD\_LENGTH + 1] = {

"", "horse", "", "hello", "harari" };

assert(cleanupRules(test2win, test2wout, 3) == 3);

assert(determineScore("horse:stable ratio is 10:1",

test2win, test2wout, TEST2\_NRULES) == 0);

//testing to see if the spaces throw the count off at all

assert(determineScore(" Happy families are all alike ; every unhappy family is unhappy in its own way . ",

test2win, test2wout, TEST2\_NRULES) == 3);

//testing to see if the spaces throw the count off at all

assert(determineScore(" Happy families are all alike ; every unhappy family is unhappy in its own way . . . ",

test2win, test2wout, TEST2\_NRULES) == 3);

//another correctness test

assert(determineScore("Happy families are all alike; every unhappy family is unhappy in its own way.",

test2win, test2wout, TEST2\_NRULES) == 3);

//testing to see if multiple unhappy two-word rules throw the count off at all

assert(determineScore("Happy horses are all alike; every unhappy horse is unhappy in its own way.",

test2win, test2wout, TEST2\_NRULES) == 2);

assert(determineScore("Happy horses are all alike; every unhappy horse is unhappy in its own way.",

test2win, test2wout, TEST2\_NRULES-1) == 1);

//another correctness test

assert(determineScore("A horse! A horse! My kingdom for a horse!",

test3win, test3wout, TEST3\_NRULES) == 1);

//correctness tests with 0 to see if there is any out of bounds accesses or errors if //there are no words inside the document

assert(determineScore("\*\*\*\* 2020 \*\*\*\*",

test3win, test3wout, TEST3\_NRULES) == 0);

assert(determineScore("",

test3win, test3wout, TEST3\_NRULES) == 0);

//testing to see if finding matches (only 1 here) is thrown off by the sheer length of //the document (251 characters)

assert(determineScore("In Yuval Noah Harari's Big data, google, and the end of free will, Harari describes the concept of Dataism, which he defines as the belief that everything can be broken down to data at a fundamental level, and that people are algorithms that process ", test3win, test3wout, TEST3\_NRULES) == 1);

//seeing if there is an error if a word is above the allowed maximum (20 characters not including zero byte)

assert(determineScore("InYuvalNoahHarari'sBigdata, google, and the end of free will, Harari describes the concept of Dataism, which he defines as the belief that everything can be broken down to data at a fundamental level, and that people are algorithms that process ", test3win, test3wout, TEST3\_NRULES) == 1);

//seeing if there is an error if a word is at the allowed maximum (20 characters not including zero byte)

assert(determineScore("InYuvalNoahHarari'sBi, google, and the end of free will, Harari describes the concept of Dataism, which he defines as the belief that everything can be broken down to data at a fundamental level, and that people are algorithms that process ", test3win, test3wout, TEST3\_NRULES) == 1);