In coding this project, I overcame major obstacles. When I initially started coding the project, I didn’t realize that an array can’t have variable length, as stated in the spec, so my whole code was contingent on a variable length array, and the numerous bugs rendered it useless. Due to this, I decided to start over. In my second attempt, I was more organized, making methods such as cleanCrib() (removes excess characters and spaces in crib) and newLineCounter() (counts number of newlines in my cipher text) that proved to be useful. Despite this, my idea was still excessively complicated and flawed. I was making a 2d array with a cipherSplit() function, which had each row of my array storing a c-string, and everytime I hit a newline character, I moved down a row in my array. This 2d array was still contingent on my new line count for the number of rows though, and thus it was still unsuccessful, although my code began to work for some test cases. When tasked once again with the issue of a variable length array, I realised that my method of decoding the ciphertext when the newline count was 0 was not contingent on any arrays of variable length, and if I reset my alphabet counters when encountering a newline character, I could successfully code a project which made the necessary replacements. Thus, I modified my decrypt, keeping most of my methods to match the cleanCrib and cipher the same. Finally, I made a checkCharMatch() function which returned true if there was no conflict when replacing the ciphertext characters with the crib characters, and printed out the final ciphertext if this was the case, or else returned false. This method finally yielded a successfully working project not contingent on arrays of variable length, and after ironing out a few bugs, I had a working project.

Another issue I had was that initially, I was fitting words into the ends of longer words in order to check if my code works, which wasn’t allowed and I thus had to later change. For example, I would put “dog bones” into a string such as “abcde fdghi” and return “abDOG BONES” and true, which is also not allowed. This was a minor issue though, and I managed to fix it quickly.

Thus, as described in the paragraphs above, other than figuring out the logic of matching the cipher and the crib, the part of the project that was the hardest for me was coding without any arrays dependent on variable length, which screwed me over in two different attempts. My final project works like this: First, I check all the preconditions, making sure the crib and cipher text were both up to par. Afterwards, I call my cleanCrib() function. This function essentially removes all the extra, unnecessary non-alpha characters in my code. It stores the new crib in a new c-string crib2. It looks something like this:

for(crib[a] doesn’t reach null byte){

if(isalpha(crib[a])){

add it to crib2,

move crib2 index up

}

else if(its not an alphabet and the previous letter was an alphabet){

add a space to crib 2

}

}

loop through and add a zero byte after last letter

Next, I called my newLineCount() function which just calculates the number of ‘\n’ in my cipher text in simple fashion. I also called my criblen() function, which is also extremely simple, counting the number of letters in each word of my crib2, essentially storing this value into an array and resetting the letter counter when reaching a space. At the end, a null byte is added to the last spot after the last number in my cribwlen array. After this, my decrypt function started becoming more complicated. This is the gist of my decrypt function:

int that traverses thru cribwlen

int that tracks alphabets in a row;

int that tracks consecutive words;

int that tracks last non-alphabet;

for(int a = 0; ciphertext[a] != '\0'; a++){

if(ciphertext[a] is newline character){

reset my counters;

}

if(its an alphabet){

increment alphabet int;

if(if consecutive alphabets = the length of word in cribwlen && its the word in crib){

store last index;

return testCharMatch(crib2, lastIndex, ciphertext, cribwlen) (diff function);

}

else if(alphabets equal and the next char isn’t an alphabet){

alph = 0;

consalph++;

trav2++;

}

else if(alphabets equal and the next char is an alphabet){

reset counters;

}

}

}

else {

if(alphabet count isn’t 0 (b/c I reset to zero if words are equal){

if(its the first word in crib){

alphabet is 0;

}

else {

everything is 0;

}

}

reset index to the last non alphabet;

}

}

return false;

In this part of my decrypt function, I call my testCharMatch() function which is what substitutes the chars into my word and tells me if it is legal to do so. (two different letters can’t be repped by same character). The testCharMatch() function looks something like this:

bool testCharMatch(cleaned crib, last index where letter found, ciphertext, cribwlen array){

int i = lastIndex;

int tc = 0; //total chars in crib

for(traversing thru criblen){

tc += cribwlen values;

}

char lig[82];

for(traversing thru ciphertext){

lig[index] = lowercase version of cipher;

}

add a null byte at the end;

int cc = 0; //going to become index of last char in a cleaner crib

char cleanestCrib[82]; //removes spaces

while(crib2 isn’t equal to the null byte’){

if(crib2 val is an alphabet)){

cleanest crib stores value

}

else{

doesn’t store

}

}

add null byte to end of cleanest crib';

cc is now index of last alpha in cleanest crib

for(int s = i; tc != 0; s--){

if(isalpha(lig[s])){

original = value of cipher (lowercase) at that pos

for(int x = 0; lig[x] != '\0'; x++){

char gai = toupper(element of cleanestCrib) (goes from end to start);

if(og is uppercase){

if(og isn’t the same as gai){

return false;

}

}

if(lig[x] == og){

change it to gai;

}

}

decrement numbers to exit loop eventually

}

}

for(int d = 0; lig[d] != '\0'; d++){

cout << values of lig;

}

So finally, after all of that my code is done and seems to work as expected for all testcases. The test cases I used were:

1. runtest("Hirdd ejsy zu drvtry od.\nO'z fodvtrry.\n", "my secret”);

works as intended, this test case returns

hiESS ejsT MY SECRET oS.

o'M foSCREET.

2) runtest("Hirdd ejsy zu drvtry od.\nO'z fodvtrry.\n", “shadow”);

works as intended, returns false b/c drvtry has repeating letters unlike shadow.

3) runtest("DiebjiggK, zyxZYXzyx--Abca abCa bdefg## $$hidbijk6437 wvuWVUwvu\n\n8 9", " hush???hUSh--- --- until JanuARY !! “);

works as intended, this test case is useful b/c it makes sure my crib is simplified as it’s supposed to be. Returns:

NATURALLY, zyxzyxzyx--HUSH HUSH UNTIL## $$JANUARY6437 wvuwvuwvu

8 9

4) runtest("boo \n boo boy boy", "dog dog”);

works as intended, returns

DOO

DOO DOG DOG

(can’t put dog in boo b/c o != g)

5) runtest("doggy", “dog”);

works as intended, returns false b/c you aren’t allowed to fit dog inside doggy.

6) runtest("hello", “");

meant to check if it works as it’s supposed to when crib is empty, returns false immediately after failing preconditions.

7) runtest("2309482", “hello");

meant to check if it works as it’s supposed to when cipher has no alphabets, returns false immediately after failing preconditions.

8)runtest("Kpio't dmpbl-boe-ebhhfs opwfm", "s cloak and”);

works as intended, an interesting check because it has a non space punctuation mark separating the letters

kOiN'S CLOAK-AND-DAhhfs NOwfL

9) runtest("boy \n boy", "dog dog”);

checks if it works when split by a newline token, returns false as it should

10) runtest("boyboy boy \n abcabc abc", "dog dog”);

initially didn’t work, because the second boy was considered dog 1 and the second dog in crib went to the second boy, but I fixed the code so now this returns false as it should.