**Name:** Eshan Uniyal **UID:** 205-172-354

# Obstacles

1. **Figuring out the algorithm**: When first approaching the problem, I thought it fairly simple and start coding without first figuring out an algorithm on paper. When it didn’t prove so easy, I was forced to take a step back and see how to break the problem down into smaller, easier subproblems, and develop incrementally.
2. **Extracting the messages**: Having never dealt with two-dimensions arrays (or C strings) before, it was tricky to write code that accurately extracts all the messages from given ciphertext while also preserving the structure of the ciphertext (with newlines, spaces, et cetera)
3. **g31 Buffer Overflow error**: I spent quite a lot of time trying to debug an AddressSanitizer error I was getting in g31 with my letters\_arr array and substitutions array, even after thoroughly checking that I wasn’t calling any index larger than 25. I later realised it was because I hadn’t appended the zero byte to those arrays, and the compiler, treating the character arrays as C strings, was unable to find a zero byte to stop moving forward in memory. Adding the zero byte to the arrays resolved the issue.

# Program Design

The program consists of five helper functions in addition to decrypt:

1. void move\_to\_back(char a[], int n, int pos): This is a function borrowed from Project 4 that pushes a character at position pos in a character array a of length n to the position after the zero byte, effectively removing it from C string
2. int detect\_match(const char a[], int n, char target): Another function borrowed from Project 4 that returns the index of the first character in a character array a of length n that matches target, or returns -1 if there’s no such match
3. void extracting\_messages(const char ciphertext[], char messages[][MAX\_MESSAGE\_LENGTH + 1], int& n\_messages): Extracts all messages from a C string ciphertext by splitting them by newline characters and inserting them in the two-dimensional array messages; modifies n\_messages to reflect the number of messages extracted.
4. void simplify\_text(char text[]): Takes an input C string text, replaces all non-letter characters with spaces, and then removes all extra spaces between the remaining words
5. bool crib\_search(const char crib[], const char message[], char substitutions[]): Checks whether C string crib matches a portion of message, modifies substitutions (an array of length 27 with the space character at each position before the zero byte) to reflect the substitutions that need to be made to decrypt the cipher, and returns true if the crib was found and false otherwise

The program utilises the following global variables:

const int MAX\_MESSAGE\_LENGTH = 81;

// max message length is 81 characters (including newline character at the end)

const int MAX\_N\_MESSAGES = 60; // 60 maximum newlines

const int N\_ALPHABET = 26; // number of letters in the alphabet

const char LETTERS\_ARR[N\_ALPHABET + 1] = "abcdefghijklmnopqrstuvwxyz";

// array of letters to compare with

Pseudocode for decrypt is as follows:

// testing for empty crib

declare Boolean words\_found to store whether crib has any words

repeatedly:

if current character in crib is a letter:

words\_found = true

break

if words\_found == false:

return false

// simplifying crib and testing its length

declare character array simple\_crib to store simplified array

call simplify\_text to simple\_crib

if simple\_crib length > MAX\_MESSAGE\_LENGTH

return false;

// extracting messages from cipher\_text, finding n\_messages

initialise two-dimensional character array messages to store all messages

call extracting\_messages to populate messages

declare character array substitutions

// searching for crib matches in each message

for each message:

call crib\_search

if crib found:

break out of loop

// returning/printing depending on whether crib was found

if crib not found:

return false

if crib found:

write every character in ciphertext with the substitution

return true

Pseudocode for extracting\_messages is as follows:

while current character in ciphertext is not zero byte:

initialise message array

repeatedly:

if current character in ciphertext is not a newline or zero byte char

add current character to message

increment index

if last character a newline character

add newline character to message

increment index

add zero byte to message

copy message into input parameter messages

Pseudocode for simplify\_text is as follows:

// replacing non-letter characters with spaces

repeatedly for array text[]:

if next character not a letter

replace with space

else

convert it to lowercase

increment character index

// removing extra spaces

repeatedly for array text[]:

if character equals space:

while next character equals space:

move next character to position after zero byte

// using move\_to\_back

// removing spaces at first or last index, if any

if space at index 0:

move space to back

if space at last index:

move space to back

Pseudocode for crib\_search is as follows:

// checking for letters in message

repeatedly:

check if character in message is a letter

increment index

if no letters found:

return false

// simplifying message

declare simple message array, copy message into it

call simplify\_text on simple message

// checking if crib shorter than simple message

if crib length > simple message length:

return false

// searching for crib in message

create array subs parallel to LETTERS\_ARR to store substitutions for each letter

append zero byte to end of subs

while true:

reset subs by filling it with spaces (other than zero byte index)

declare bool match\_found = true

while true:

if current crib and current message characters are both letters:

if no known substitution exists:

if substitution has already been assigned to a letter:

match\_found = false

break

else:

assign substitution to current character in subs

else:

if known substitution does not match message character:

match\_found = false

break

if not both are spaces:

match\_found = false

break

increment crib index and simple message index

if crib ran out:

if current word in simple message hasn’t run out:

match\_found = false

break

else if crib hasn’t run out but simple message has:

match\_found = false

break

if match\_found == true:

return true

else:

check if there’s a next word in simple message

if next word found:

update start index, outer loop runs again to search for crib with a new start index

if no next word:

no match found, return false

# Tests

const char crib\_1[33] = "hush-hush until January 20, 2021";

const char cipher\_1[27] = "Vlgmgak zfefyfektkyy.\nIfk!";

assert(decrypt(cipher\_1, crib\_1) == false);

// testing a crib with non-letter characters against an ordinary ciphertext

// shouldn’t write anything

const char cipher\_2[73] = "DiebjiggK, zyxZYXzyx--Abca abCa bdefg## $$hidbijk6437 wvuWVUwvu\n\n8 9\n";

const char crib\_2[48] = " hush???hUSh--- --- until JanuARY !! ";

assert(decrypt(cipher\_2, crib\_2) == true);

// testing a cipher with non letter characters and extra spacing with a cipher it should match

// writes:

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

8 9

const char cipher\_3[184] = "Zysqjs zbguncyqzo jdsbyo eybmnu bg Wqzsvbbf.\nUnysqx eybmgxrsuu ymtbyu kcq Jicjjsy.\nNbuj sajysts rcvsyqr qgx sajysts zbgusykqjcks nbucjcbgu bg xcuzmuucbg wbymtu.\nZU 31 cu zdqrrsGECGE!";

const char crib\_3[19] = "conspiracy THEORY";

assert(decrypt(cipher\_3, crib\_3) == true);

// testing a multiline cipher with regular characters with a crib with regular characters

// writes:

CREATE CONSPIRACY THEORY eROmPS ON wACEvOOf.

SPREAx eROmNxrESS RmtORS kIA TiITTER.

POST EaTREtE rIvERAr ANx EaTREtE CONSERkATIkE POSITIONS ON xISCmSSION wORmtS.

CS 31 IS CHArrENeINe!

const char cipher\_4[45] = "abcd efg hijk ilm nppq rst dccb cba uwwx wyz";

const char crib\_4[9] = "good one";

assert(decrypt(cipher\_4, crib\_4) == true);

// testing a one-line cipher with multiple possible matches (in terms of length), but only one valid match

// writes:

abcd efg hijk ilm nppq rst dccb cba GOOD ONES

const char cipher\_5[40] = "Hirdd ejsy zu drvtry od.\nO'z fodvtrry.\n";

const char crib\_5[10] = "my secret";

assert(decrypt(cipher\_5, crib\_5) == true);

// testing another regular multiline cipher with regular characters with a crib with regular characters

// writes:

hiESS ejsT MY SECRET oS.

o'M foSCREET.

const char crib\_6[7] = "shadow";

assert(decrypt(cipher\_5, crib\_6) == false);

// testing a regular multiline cipher with regular characters with a crib that doesn’t match

const char cipher\_6[6] = "28\n39";

assert(decrypt(cipher\_6, crib\_6) == false);

// testing a multiline cipher with no letter characters

const char cipher\_7[1] = "";

assert(decrypt(cipher\_7, crib\_6) == false);

// testing an empty cipher

const char cipher\_8[30] = "Kpio't dmpbl-boe-ebhhfs opwfm";

const char crib\_7[12] = "s cloak and";

assert(decrypt(cipher\_8, crib\_7) == true);

// testing a one-line cipher with a regular crib

const char crib\_8[1] = "";

assert(decrypt(cipher\_8, crib\_8) == false);

// testing empty crib

const char crib\_9[8] = "273\n823";

assert(decrypt(cipher\_8, crib\_9) == false);

// testing crib with only non-letter characters

const char cipher\_9[5] = "y es";

assert(decrypt(cipher\_9, crib\_7) == false);

// testing cipher shorter than crib