Kelly Yu

Dr. Smallberg

CS 31

November 14, 2021

**Project 5: Secret Plans**

**A. Notable Obstacles**

A notable obstacle I am still working on overcoming is not feeling overwhelmed by the number of details. I need to grasp the bigger picture and do a rough sketch of the major parts that make up the picture. I think this will help me feel more confident, because I know that each piece of code I am building will likely get me closer to building the whole project.

Something that did help was laying out my ideas for each section of the code before actually implementing it. When checking if the pattern of repeated characters matched (after confirming the right character length), I noticed a flaw in my initial idea. Had I not laid out the idea on paper and just typed the code, I would have had to discard most of it and probably spend a large amount of time debugging code that I wouldn’t even use later. However, I do want to balance between laying out ideas and implementing them. I don’t want to draw a full-scale idea on paper before I implement it, because sometimes having the compiler check the functionality of my ideas may help.

Also, I must try to be more cautious about my test cases. I realized that I sometimes don’t even know what the results should be, so it’s funny how I can even evaluate whether my code is working. I am practicing using the debugging tools and breakpoints to check the values of my variables. This helped me figure out why my textPositions array was storing sequentially correct but wrong valued element numbers.

Another common obstacle I face is knowing which of my closing brackets corresponds to which of my opening brackets. This tends to happen when I have long for loops with nested if-statements and nested loops. I am now using comments to mark the closing brackets before I begin adding to the body statements. I am also practicing using functions to replace these long chains nested code.

**B. Pseudocode**

declare function prototypes();

main function()

…

decrypt function()

find number of (interesting) characters in crib

if num chars is 0 or > 90

return false because crib is never found in ciphertext

//otherwise

//reformat copy of the crib

create a counter to track num of characters we copy

create char array to store reformatted copy of the crib

go through chars in crib and copy characters to reformatted copy of crib according to this format:

if there are two consecutive letters,

set first letter to lowercase and store it

else if letter is followed by null character

set letter to lowercase, and store the letter and null character

copying is finished

else if letter is followed by a non-letter

store the letter and the space

if last character stored is a space

change the last character to a null character to end the array

//reformat copy of ciphertext

find number of (interesting) characters in ciphertext

create a counter to track num of characters we copy

create char array to store reformatted copy of ciphertext

go through chars in ciphertext and copy characters to reformatted copy of ciphertext according to this format:

if char in ciphertext is a letter

lowercase the letter and store it

else if char in ciphertext is a null character

store the null character

else if non-letter char is followed by a letter char

store a space

assign null character as the last character in reformatted ciphertext to end the char array

//checking for RIGHT LENGTH of reformatted crib in reformatted ciphertext

create integers cribPos and textPos to store positions of crib and ciphertext elements we observe

create counter for number of character types that match

create integer startIndex to set the point where we start traversing through reformatted ciphertext

create integer to store 1st element position where crib is “found” in reformatted ciphertext

create Boolean matchFound that’s false until actual match (right length and patten) is found

determine num of interesting characters in reformatted crib

for every interesting character in reformatted ciphertext{

if elements at corresponding positions in reformatted crib and ciphertext are both letters

add 1 to number of chars matched

move to next element in reformatted crib

else if elements at corresponding positions in reformatted crib and ciphertext are both spaces

add 1 to number of chars matched

move to next element in reformatted crib

else

reset number of chars matched to 0

reset search for reformatted crib all over again (element 0)

add 1 to our starting point in reformatted ciphertext

assign new starting point to position (textPos) in reformatted ciphertext we observe next

subtract 1 from textPos since it will increment by 1 after this else body (offset increment)

if num of chars matched equals num interesting chars in reformatted crib,

further check if element after the matched section in reformatted ciphertext is a non-char{

store the starting position of the match in reformatted ciphertext

if match begins in the middle of ciphertext AND

if element before the matched section in is a letter (not a match)

reset number of chars matched to 0

reset search for reformatted crib all over again (element 0)

add 1 to our starting point in reformatted ciphertext

assign new starting point to position (textPos) in reformatted ciphertext we observe next

subtract 1 from textPos since it will increment by 1 after this else body (offset increment)

//otherwise

else if checkPattern returns true //reformatted crib & matched section have same repeated pattern chars

set Boolean matchFound to true

stop further checking for reformatted crib in reformatted ciphertext

else

reset number of chars matched to 0

reset search for reformatted crib all over again (element 0)

add 1 to our starting point in reformatted ciphertext

assign new starting point to position (textPos) in reformatted ciphertext we observe next

subtract 1 from textPos since it will increment by 1 after this else body (offset increment)

} //outer if

}//outer for loop

create copy of original ciphertext with all letters lowercased & non-letters + null char maintained

if actual match is found{

for every character in reformatted crib up to null char{

if character is a letter{

for every character in copy of original ciphertext up to null char{

if character is the corresponding reformatted ciphertext’s letter{

change the character to the reformatted crib’s letter

}

}

}

}

}

if actual match never found //no section of reformatted ciphertext matches length and pattern of reformatted crib

return false

//otherwise

print out the (partially) decrypted message

return true

-----

checkPattern function()

/\* assuming we passed these arguments:

reformatted crib, reformatted cyphertext, and element position of where the match starts \*/

determine num of interesting chars in reformatted crib

create integer firstAppear to store where a specified letter first appears in reformatted crib

create integer array1 to store the position numbers of specified letter found in reformatted crib

create integer array2 to store position numbers of letter corresponding to specified letter found in reformatted ciphertext

for each letter in the alphabet{

set firstAppear to -1 //prepares search of next letter in reformatted crib

go through every char in reformatted crib until we get first element position where letter is found

assign the element position where letter is first found to firstAppear

stop checking for letter in the crib

if firstAppear is not -1 anymore{ // cribLetter was found in reformatted crib

create a counter for num of interesting integers we store //for reformatted crib

go through every character in reformatted crib

if letter is found in crib

store the element position into integer array1

add 1 to the counter

retrieve ciphertext letter whose element position corresponds to letter’s position in reformatted crib

create another counter for num interesting integers we store //for reformatted ciphertext

go through every character in reformatted ciphertext

if corresponding ciphertext letter is found in reformatted ciphertext

store element position corresponding to element position in reformatted crib into integer array2

add 1 to the counter

//check if repeated pattern in reformatted crib exists in reformatted ciphertext

if number of element positions stored in array1 and array2 are different

return false; //repeated pattern does not exist in ciphertext

for every integer in array1 and array2

if array1 and array2 are not identical

return false; //repeated pattern does not exist in ciphertext

} //if statement

} //outer for loop

return true; //repeated pattern in reformatted crib exists in reformatted ciphertext

**C. Test Cases**

using the function:

**void** runtest(**const** **char** ciphertext[], **const** **char** crib[]){

cout << "====== " << crib << endl;

**bool** result = decrypt(ciphertext, crib);

cout << "Return value: " << result << endl;

}

and including this in main()

cout.setf(ios::boolalpha); // output bools as "true"/"false"

|  |  |
| --- | --- |
| Ciphertext contains no letters 🡪 nothing to decrypt; crib never found in ciphertext  **====== hi**  **Return value: false** | runtest("1273618329", "hi"); |
| Ciphertext contains nothing 🡪 nothing to decrypt  **======**  **Return value: false** | runtest("", ""); |
| Crib contains no letters 🡪 no key can be made  **======**  **Return value: false** | runtest("hello", ""); |
| Crib contains more than 90 characters 🡪 never found in ciphertext, nothing decrypted  **====== hellogood1hellogood1hellogood1hellogood1hellogood1hellogood1hellogood1hellogood1hellogood1hellogood1**  **Return value: false** | runtest("hello", "hellogood1hellogood1hellogood1hellogood1hellogood1hellogood1hellogood1hellogood1hellogood1hellogood1"); |
| Crib & ciphertext have a phrase with same length and same repeated character pattern 🡪 decryption occurs  -- **ciphertext has no newlines, ends with null char**  **====== eat**  **EATReturn value: true** | runtest("hop", "eat"); |
| Crib & ciphertext have a phrase with same length and same repeated character pattern 🡪 decryption occurs  -- **ciphertext’s last message ends with newline, followed by null character**  **====== eat**  **EAT**  **Return value: true** | runtest("hop\n", "eat"); |
| Crib & ciphertext have a phrase with same length and same repeated character pattern 🡪 decryption occurs  -- **ciphertext’s last message starts and ends with newline; fully ends with null character**  **====== eat**  **EATReturn value: true** | runtest("\nhop\n", "eat"); |
| Crib & ciphertext have a phrase with same length and same repeated character pattern 🡪 decryption occurs  -- **ciphertext’s last message starts with newline & ends with null character**  **====== eat**  **EAT**  **Return value: true** | runtest("\nhop", "eat"); |
| Multiple sections in ciphertext that fully match crib; use the first section to make the key  **====== eat**  **EAT**  **EiTReturn value: true** | runtest("\nhop\nhip", "eat"); |
| ‘odp’ is split between a newline (in different messages); crib matches the length and pattern, but must be found in one message, not two  **====== eat**  **o**  **dT EATReturn value: true** | runtest("o\ndp hip", "eat"); |
| ‘tickle’ == 6-letter word, not two 3-letter words 🡪 crib is not found in ciphertext  --ciphertext section has right length, but if section is in between ciphertext, the character before the start of the ciphertext should not be a letter to indicate start of a new word;  --must check for non-letters before and after section  **====== eat**  **Return value: false** | runtest("pickbd", "eat"); |
| Original ciphertext has uppercase letters, but if they are not all decrypted, only those decrypted are uppercase  **====== eat**  **EAT ky**  **Return value: true** | runtest("PIC KY\n", "eat"); |
| Original ciphertext’s non-letters are maintained even after decryption  **====== eat**  **EAT ... ky**  **Return value: true** | runtest("PIC ... KY\n", "eat"); |
| Plaintext letters can replace itself – a matched with a  **====== eat**  **EATReturn value: true** | runtest("PAC", "eat"); |
| Plaintext letters cannot be replaced by more than one ciphertext – 1 to 1 match  **====== ehtt**  **Return value: false** | runtest("PACA", "ehtt"); |
| Uppercase ciphertext does not affect whether crib is found in ciphertext  **====== pot**  **POTReturn value: true** | runtest("hEy", "pot"); |
| Right length & right repeated pattern 🡪 decryption occurs  **====== aegh . pt**  **yeHE AEGH 123 PTReturn value: true** | runtest("yelo uoil 123 bc", "aegh . pt"); |
| Right length but not right repeated patten 🡪 no decryption  **====== aegh . pt**  **Return value: false** | runtest("yelo uoiu 123 bc", "aegh . pt"); |
| ‘uoiu bc’ == right length, wrong pattern  goes on to analyze ‘loqi yn’ == right length & pattern  **====== aegh . pt**  **PeAE uEHu 123 bc AEGH \_\_ PTReturn value: true** | runtest("yelo uoiu 123 bc loqi \_\_ yn", "aegh . pt"); |
| right length & right pattern (‘ll’ matches ‘hh’)  **====== aeghh . pt**  **yeHE AEGHH 123 PTReturn value: true** | runtest("yelo uoill 123 bc", "aeghh . pt"); |
| Crib ends with non-letters – reformatted correctly  **====== aeghh . pt...**  **yeHE AEGHH 123 PTReturn value: true** | runtest("yelo uoill 123 bc", "aeghh . pt..."); |
| Crib with multiple non-letters in between – reformatted correctly  **====== aeghh \_\_\_ pt**  **yeHE AEGHH 123 PTReturn value: true** | runtest("yelo uoill 123 bc", "aeghh \_\_\_ pt"); |
| Crib with 1 non-letter in between – reformatted correctly  **====== aeghh \_\_\_ pt**  **yeHE AEGHH 123 PTReturn value: true** | runtest("yelo uoill 123 bc", "aeghh pt"); |
| ciphertext with multiple non-letters in between – reformatted correctly  **====== aegh**  **AEGH 00 uoReturn value: true** | runtest("yeln 00 uo", "aegh"); |
| ciphertext with 1 non-letter in between – reformatted correctly  **====== aegh**  **AEGH 00 uoReturn value: true** | runtest("yeln uo", "aegh"); |
| ciphertext ends with non-letters – reformatted correctly  **====== aegh**  **AEGH uo ... Return value: true** | runtest("yeln uo ... ", "aegh"); |
| Multiple repeated letters match up  **====== aeggh tt**  **AEGGH TT ... Return value: true** | runtest("yelln uu ... ", "aeggh tt"); |