**Notable Obstacles:**

1. The first obstacle comes from a trouble on normalizing the crib to the standard format, that is, consisting of only words and blanks between each two words. For blanks or other signs and marks before the first word, it’s sort of simple to ignore them. However, it really took me some time to consider how to get rid of blanks or other signs and marks after the last word. In fact, assigning the character just after the last word to ‘\0’ can easily solve this problem.
2. In the lecture, we have learnt to use *getline* or *get* to assign values to the c-string. However, these methods make me wonder can’t we assign the value simple as that we do for array? The answer is yes.

char a[] = "d s a ";

char b[200];

for (int i = 0; i < 8; i++)

{

b[i] = a[i];

}

cout << b;

This works well for c-string.

1. To test whether a crib exists in a ciphered text or not, I need to not only compare the lengths, but also check if any two or more same letters in ciphered text correspond to two or more same letters in crib. It’s relatively simple to compare the lengths. As for letters correspondences, I defined a function with double for-loops.
2. For most situations, I cannot find the crib in the first several lines. Therefore, I have to save all the lines in the file so that once I find a cipher, I can decrypt all messages required. To achieve that, I created two 2-dimensional c-strings. One of them keeps all the lines of the file and the other keeps all the lines after normalizing of the file.
3. For a c-string a[] with strlen(a) = r, the syntax to create another c-string with the same size should be char b[strlen(a) + 1] so that ‘\0’ can be filled in this c-string.
4. During the process of debugging, I found that the output was the correct plaintext followed by several messy codes like: \205\377\377. After several tests, I found that blank lines generated these messy codes. That is, since I just used (int f = 0; f = 51; f++) to scan all the lines, there could be only 30 lines containing words. For the remaining 21 lines, I will receive bad results like the messy codes. Therefore, I counted the number of lines while I was storing all lines into a 2-dimensional c-string. And then, I used (int f = 0; f < num; f++) to scan all the lines and decrypt them one by one.

**Description/Pseudocode:**

1. First of all, the program should normalize the crib, which may contain a lot of useless characters for decryption.

2. Then, normalize lines (delete useless characters) read from the file so that we may directly and subtly find the characters in the ciphertext that correspond to those in the crib.

3. Check from the first character in the normalized ciphertext, if we are supposed to find the code in ciphertext, the length should be one of the key points.

4. Even though the formats are the same, it’s not guaranteed that same letters in the crib represent the same letters in the ciphertext. Create a c-string cipher which holds the corresponding characters of the crib in ciphertext. Use isConvertible to check whether it has the same-letter problem.

5. If all ready, convert the ciphertext into plaintext. If meeting characters other than letters, just copy them. Otherwise, find the corresponding letter in the crib and assign the letter with appropriate case to appropriate position in the plaintext.

void truncate:

repeatedly from the end:

if the character is a blank, then continue to check the one before

if the character is not a blank, assign ‘\0’ to the character right after this one, marking the end of the c-string.

bool isConvertible:

for two c-strings with the same size, repeatedly:

compare each two elements with the same index in two c-strings

continue to compare if both elements are blanks or letters

return true after the whole comparing process

return false as soon as the program detects two elements that so not satisfy the conditions

bool decrypt:

process crib[]:

create a new c-string to place the normalized crib

repeatedly:

if the element of crib is a letter, copy that and assign to the new c-string

else if it is a weird character,

if there are some consecutive weird characters, assign a blank to appropriate position in the new c-string for only one time

else assign a blank

after these steps, a new crib forms

return false if the new crib contains no words

call truncate to get rid of possible blanks after the last letter character in new crib(newCrib[])

process cipherstream as the method to process crib[](newCipherText):

after that, if the length of newCrib[] is larger than that of newCipherText, go to check the next line read from the file

repeatedly(k increment):

repeatedly(n increment, k constant):

if ith element in crib and (k+i)ith element are both letters or blanks, continue to check

else k increment

jump out the loop and check next line read from the file

if all elements satisfy the condition, create a c-string cipher

if cipher and newCrib is convertible:

create a c-string decrypted to hold the plaintext

for each space in decrypted, assign the value according to different situations:

(1). The character in ciphertext is not a letter:

just copy that

(2). The character in ciphertext is a letter and can be found in cipher:

assign the lowercase of the corresponding letter in the crib to that space

(3). The character in ciphertext is a letter and cannot be found in cipher:

leave the uppercase of the corresponding letter in the crib to that space

if cipher and newCrib is not convertible(due to same-letter problem):

jump out of the loop and find the cipher again

return false if we never find that cipher

**Test Cases:**

void runtest(const char filename[], const char crib[])

{

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

ifstream cfile(filename);

if(!cfile)

{

cout << "Cannot open " << filename << endl;

return;

}

bool result = decrypt(cfile, crib);

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

}

1. **test data.txt:**

Tftdn lxwdqkoft lhgzztr of Ltezgk zvtfzn-ltctf.

O ziofa zit Zktqlxkt gy Doeiossofrq ol wxkotr lgdtvitkt of dn wqeanqkr.

Ror ngx itqk ziqz Sgktzzq ol ugofu gxz vozi Dqkexl?

1. runtest(“test data.txt”, “ “):

======

Return value: false

1. runtest(“test data.txt”, “!^@ &#% $”):

====== !^@ &#%$

Return value: false

1. runtest(“test data.txt”, “3271463”):

====== 3271463

Return value: false

1. runtest(“test data.txt”, “ &\*323167( “):

====== &\*323167(

Return value: false

1. runtest(“test data.txt”, “treasure of michillinda”):

====== treasure of michillinda

enemN suWmarine sHotted in sector tVentN-seCen.

i thinA the treasure of michillinda is Wuried someVhere in mN WacANard.

did Nou hear that loretta is UoinU out Vith marcus?

Return value: true

1. runtest(“test data.txt”, “%237abcbd\*”):

====== %237abcbd\*

bdbDN aXWDQKOdb aHGZZbR Od abEZGK ZVbdZN-abcbd.

O ZIOdA ZIb ZKbQaXKb GY DOEIOSSOdRQ Oa WXKObR aGDbVIbKb Od DN WQEANQKR.

ROR NGX IbQK ZIQZ SGKbZZQ Oa UGOdU GXZ VOZI DQKEXa?

Return value: true

1. runtest(“test data.txt”, “ ded”):

====== ded

Return value: false

1. **test data.txt:**

Tftdn lxw^!##dqkoft lhgzztr of Ltezgk zvtfzn-ltctf.

O ziofa zit Zktqlxkt @#432gy Doeiosso@#43frq ol wxkotr lgdtvitkt of dn wqeanqkr.

Ror ngx itqk zi12@qz Sgktzzq ol ugo@3fu gxz vozi Dqkexl?

1. runtest(“test data.txt”, “ded”):

====== ded

Return value: false

1. runtest(“test data.txt”, “abcbd”):

====== abcbd

bdbDN aXW^!##DQKOdb aHGZZbR Od abEZGK ZVbdZN-abcbd.

O ZIOdA ZIb ZKbQaXKb @#432GY DOEIOSSO@#43dRQ Oa WXKObR aGDbVIbKb Od DN WQEANQKR.

ROR NGX IbQK ZI12@QZ SGKbZZQ Oa UGO@3dU GXZ VOZI DQKEXa?

Return value: true

1. runtest(“test data.txt”, “@#4pcdurcfa@ “

====== @#4pcdurcfa@

TFTDr LXp^!##DcfOFT LHGZZTa OF LTdZGf ZVTFZr-LTCTF.

O ZIOFu ZIT ZfTcLXfT @#432GY DOdIOSSO@#43Fac OL pXfOTa LGDTVITfT OF Dr pcdurcfa.

aOa rGX ITcf ZI12@cZ SGfTZZc OL UGO@3FU GXZ VOZI DcfdXL?

Return value: true

1. runtest(“test data.txt”, “woA^@37DSUx”):

====== woA^@37DSUx

sFsDw LaW^!##DuxOFs LHoZZsR OF LsEZox ZVsFZw-LsCsF.

O ZdOFA Zds ZxsuLaxs @#432oY DOEdOSSO@#43FRu OL WaxOsR LoDsVdsxs OF Dw WuEAwuxR.

ROR woa dsux Zd12@uZ SoxsZZu OL UoO@3FU oaZ VOZd DuxEaL?

Return value: true

1. runtest(“test data.txt”, “!@332da”):

====== !@332da

TaTDN LXW^!##DQKdaT LHGZZTR da LTEZGK ZVTaZN-LTCTa.

d ZIdaA ZIT ZKTQLXKT @#432GY DdEIdSSd@#43aRQ dL WXKdTR LGDTVITKT da DN WQEANQKR.

RdR NGX ITQK ZI12@QZ SGKTZZQ dL UGd@3aU GXZ VdZI DQKEXL?

Return value: true