1. The notable obstacle I encountered is to find the position that the new ciphertext corresponds with the new crib. Later, I tried to break the obstacle into three sections, firstly, check if the ciphertext and crib have the same structure; secondly, if they have same structure, check if neighboring of the corresponding ciphertext is not alphabet; thirdly, check if the alphabets are corresponding. After separating one overwhelmed problem into 3 parts, the problem got solved.

2.

If crib has no words,

Return false

Repeatedly:

Count the length of valid crib

If the char of crib is alphabet,

Increment the length of valid crib

If the char of crib is not alphabet and the next char is alphabet,

Increment the length of valid crib

If the first index of crib is not alphabet,

Reduce length of valid crib by 1

If the length of valid crib is greater than 90 or less than 0,

Return false

Repeatedly:

Eliminate non-letter char but keep necessary space for crib

If the char of original crib is alphabet,

Add the original char in uppercase form to the new crib

If the char of original crib is not alphabet, the next char is alphabet, and the char is not in the first place,

Replace the char with space

Repeatedly:

Eliminate non-letter char but keep necessary space for ciphertext

If the char of original ciphertext is alphabet or \n,

Add the original char in upper case form to the new ciphertext

If the char of original ciphertext is not alphabet, the next char is alphabet

Replace the char with space

Repeatedly:

Compare new crib with new ciphertext

Check if there exist same structure

Check if neighboring chars of the structure are space

Check if alphabets are corresponding

If any condition from above does not satisfy,

Return false

Repeatedly:

Replace chars in original ciphertext with existing corresponding crib letter, and change other chars in lowercase form

Print every char of the plaintext letters

3.

**const** **char** a[1000]="F gspt fe! zyxZYXzyx--Abca abCa bdefg## $$dsptrqtj6437 wvuWVUwvu\n\n8 9\n";

**const** **char** b[100]="";

decrypt(a,b);

In this case, the output will be nothing, but if I write “cout<<decrypt(a,b);”, it will cout 0, which I used to test crib with non-letter. If I did not

**const** **char** a[1000]="F gspt ff! zyxZYXzyx--Abca abCa bdefg## $$dsptrqtj6437 wvuWVUwvu\n\n8 9\n";

**const** **char** b[100]="AA";

decrypt(a,b);

In this case, check if the corresponding char become uppercase and other char become lowercase.

**const** **char** a[1000]="F gspt ff! zyxZYXzyx--Abca abCa bdefg## $$dsptrqtj6437 wvuWVUwvu\n\n8 9\n";

**const** **char** b[100]="22";

decrypt(a,b);

In this case, the output will be nothing, but if I write “cout<<decrypt(a,b);”, it will cout 0, which I used to test if crib is number. At first, I did not write if (validLength<=0){

return false;}, then the function output the original ciphertext, which is not what we want.

**const** **char** a[1000]="F gspt ff! zyxZYXzyx--Abca abCa bdefg## $$dsptrqtj6437 efg\n\nhehe 9\n";

**const** **char** b[100]="abc\n\n8hehe";

decrypt(a,b);

In this case, the ouput will be nothing, but if I write “cout<<decrypt(a,b);”, it will cout 0, which I used to test if the crib will mistakenly match the corresponding ciphertext, which it is not what we want, since we should regard next line as a new begin.