Perform encryption and decryption using the following transposition techniques

1. Rail fence - Row & Column Transformation
2. One time pad

A **rail fence cipher** is one in which plaintext symbols are rearranged (i.e., transposed or permuted) to produce ciphertext. The method of transposition may be either mathematical or typographical in nature

3.1 Columnar Transposition:

The number of columns is the key information.

To encipher : Plaintext is written horizontally in k columns, and is then transcribed vertically columnby-column,

To decipher : Suppose that the length of the ciphertext is n and the key is k. Then the letters will fill n DIV k full rows, and there will be one partial row at the end with n MOD k letters. Transcribing row-by-row will then yield the plaintext.
Example: Encrypt NOTHING IN THE WORLD IS MORE DANGEROUS THAN SINCERE IGNORANCE AND CONSCIENTIOUS STUPIDITY with a key of k = 9 columns. Solution: We write the plaintext horizontally in 9 columns as follows:

N	O	T	H	I	N	G	I	N
T	H	E	W	O	R	L	D	I
S	M	O	R	E	D	A	N	G
E	R	O	U	S	T	H	A	N
S	I	N	C	E	R	E	I	G
N	O	R	A	N	C	E	A	N
D	C	O	N	S	C	I	E	N
T	I	O	U	S	S	T	U	P
I	D	I	T	Y				

The cipher text is therefore:
NTSESNDTIOHMRIOCIDTEOONROOIHWRUCANUTIOESENSSYNRDTRCCSGLAHEEITIDNAIAEU NIGNGNNP.

Example: Suppose the ciphertext is: GPSDO AILTI VRVAA WETEC NITHM EDLHE TALEA ONME. If it is known that the key is k = 7, find the plaintext. Solution: There are 39 letters in the ciphertext which means that there are 39 DIV 7 = 5 full rows and one partial row with 39 MOD 7 = 4 letters

				×	×	×

➢ **METHODOLOGY FOLLOWED:**

```cpp
#include <iostream>
#include<bits/stdc++.h>

using namespace std;

int main()
{
   ifstream fin;
   fin.open("input.txt") ;
   string st;


   string keyst;
   getline(fin,keyst);


    int key=0;
    int sz1=keyst.size();
    int p=1;

    for(int i=sz1-1;i>=0;i--){
        key+= (keyst[i]-'0')*p;
        p*=10;
    }

    cout<<key<<"\n";



  /* if(keyst.size()==1){
       key=keyst[0]-'0';
   }
   else{
       key= (keyst[0]-'0')*10 + (keyst[1]-'0');
   }*/

   // getline(fin,st);
   // int n=st.size();
   // cout<<st<<"\n\n";


  // vector<int> space;
   vector< vector<char> > vf;

   int c=0;
   vector<char> vc;
   char ch;
   int column=0;
```

```
 cout<<"\n**** plain text is ****\n\n";
while(fin.get(ch)){
    cout<<ch;
    vc.push_back(ch);
    c++;
    if(c==key){
      vf.push_back(vc);
      vc.clear();
      c=0;
    }
}


 if(vc.size()!=0){
     while(c<key){
         vc.push_back(' ');
         c++;
     }
    vf.push_back(vc);
 }

 cout<<"\n**** plain text matrix is ****\n\n";
 for(vector<char> v : vf){
      for(char ch: v){
          cout<<ch<<" ";
      }
      cout<<"\n";
 }
 cout<<"\n";

 string encrypted;

 int sz = vf.size();


 for(int i=0;i<key;i++){
    for(int j=0;j<sz;j++){
       /* if(vf[j][i]!=' '){
          encripted.push_back(vf[j][i]);
       } */
       encrypted.push_back(vf[j][i]);
  }

 }
 cout<<"****  encrypted message is: ****\n\n";
 cout<<encrypted<<"\n";
 ofstream fout1;
 fout1.open("output.txt") ;
 fout1<<encrypted<<"\n";
```

```
vc.clear();
vf.clear();
c=0;
int n = encrypted.size() ;
int row = n/key ;
if(n%key!=0){
 row++;
}

char matrix[row][key];

if(n%key!=0){
   int r = n%key;

   for(int i=r;i<key;i++){
     matrix[row-1][i]='*';
   }
}

 int k=0;
 string decrypted="";

 for(int i=0;i<key;i++){
     for(int j=0;j<row;j++){
         if(matrix[j][i]!='*'){
             matrix[j][i]=encrypted[k];
             k++;
         }
     }
 }

 for(int i=0;i<row-1;i++){
     for(int j=0;j<key;j++){

      decrypted.push_back(matrix[i][j]);
     }

 }

  for(int j=0;j<key;j++){
     if( matrix[row-1][j] !='*' ) decrypted.push_back(matrix[row-1][j]);
  }
```

```
    cout<<"\n****  decrypted message is: ****\n\n";
    cout<<"\n"<<decrypted<<"\n";

    ofstream fout2;
    fout2.open("doutput.txt");

    fout2<<decrypted;
    fout2.close();

    fin.close();
    fout1.close();




    return 0;
}
```

INPUT:

INPUT FILE: (PLAIN TEXT WITH KEY)

main.cpp | input.txt

input.txt

```
1   9
2   NOTHING IN THE WORLD IS MORE DANGEROUS THAN SINCERE IGNORANCE AND CONSCIENTIOUS STUPIDITY !!!
3
```

OUTPUT:

OUTPUT FILE: (ENCRYPTED TEXT)

output.txt

```
1   NERTI S!O EHGC !TW ANOS
2   HODNONT
3   IRA RSU
4   NLNSACP GDGINII   ENCED IIRCENI NSOE TT   URAIY TMSENO  HO  DU!
5
```

DOUTPUT FILE: (DECRYPTED TEXT)

doutput.txt

```
1   NOTHING IN THE WORLD IS MORE DANGEROUS THAN SINCERE IGNORANCE AND CONSCIENTIOUS STUPIDITY !!!
2
```

TERMINAL OUTPUT:

```
key: 9
**** plain text is ****
NOTHING IN THE WORLD IS MORE DANGEROUS THAN SINCERE IGNORANCE AND CONSCIENTIOUS STUPIDITY !!!


**** plain text matrix is ****
N O T H I N G   I
N   T H E   W O R
L D   I S   M O R
E   D A N G E R O
U S   T H A N   S
I N C E R E   I G
N O R A N C E   A
N D   C O N S C I
E N T I O U S   S
T U P I D I T Y
! ! !



****  encrypted message is: ****
NNLEUINNET!O D SNODNU!TT D CR TP!HHIATEACII
IESNHRNOOD
N  GAECNUI
GWMEN ESST  OOR I C Y IRROSGAIS
****  decrypted message is: ****

NOTHING IN THE WORLD IS MORE DANGEROUS THAN SINCERE IGNORANCE AND CONSCIENTIOUS STUPIDITY !!!
```

3.2 Keyword Columnar Transposition

The order of transcription of the columns is determined by the alphabetical order of letters in the keyword. If there are repeated letters in the keyword, the columns corresponding to those letters are transcribed in order left-to-right

Example: Encrypt THE QUICK BROWN FOX JUMPED OVER THE LAZY DOG if the keyword is CORNELL.

C	O	R	N	E	L	L
1	6	7	5	2	3	4
T	H	E	Q	U	I	C
K	B	R	O	W	N	F
O	X	J	U	M	P	E
D	O	V	E	R	T	H
E	L	A	Z	Y	D	O
G						

Ciphertext: TKODE GUWMR YINPT DCFEH OQOUE ZHBXO LERJV A

- **METHODOLOGY FOLLOWED:**

```cpp
#include <iostream>
#include<bits/stdc++.h>

using namespace std;


int main()
{
   ifstream fin;
   fin.open("input.txt") ;
   string st;

   string keyword;
   getline(fin,keyword);

   cout<<"key word:"<<keyword<<"\n";
   int key=keyword.size();
   cout<<key<<"\n";

   map<char,vector<int>>m;
   unordered_map<char,vector<int>> m1;
   for(int i=0;i<key;i++){
     m1[keyword[i]].push_back(i);
   }


   string temp = keyword;
   cout<<temp<<"\n";
   int a[key];
   sort(temp.begin(),temp.end());
   cout<<temp<<"\n";
   for(int i=0;i<key;i++){
     m[temp[i]].push_back(i);
     cout<<temp[i]<<" "<<i<<"\n";
   }

   vector< vector<char> > vf;

   int c=0;
   vector<char> vc;
   char ch;
   int column=0;
   cout<<"\n**** plain text is ****\n\n";

  while(fin.get(ch)){
      cout<<ch;
      vc.push_back(ch);
      c++;
      if(c==key){
```

```
      vf.push_back(vc);
      vc.clear();
      c=0;
    }
}


 if(vc.size()!=0){
     while(c<key){
         vc.push_back(' ');
         c++;
     }
    vf.push_back(vc);
 }

 cout<<"\n**** plain text matrix is ****\n\n";

 for(vector<char> v : vf){
      for(char ch: v){
          cout<<ch<<" ";
      }
      cout<<"\n";
 }
 cout<<"\n";

 string encrypted;

 int sz = vf.size();

 map<char,vector<int>> :: iterator it=m.begin();

 for(it=m.begin();it!=m.end();it++){

      for(int p:m1[it->first]){
          for(int j=0;j<sz;j++){

        encrypted.push_back(vf[j][p]);
        }
      }
 }


 cout<<"****  encrypted message is: ****\n\n";

 ofstream fout1;
 fout1.open("output.txt") ;
 fout1<<encrypted<<"\n";
```

```
cout<<encrypted<<"\n";

vc.clear();
vf.clear();

//decryption ----------------------------------------------------------------


c=0;
int n = encrypted.size() ;
int row = n/key ;
if(n%key!=0){
 row++;
}

char matrix[row][key];

 int k=0;
 for(int i=0;i<key;i++){
     for(int j=0;j<row;j++){
             matrix[j][i]=encrypted[k];
             k++;
         }
 }

 for(int i=0;i<row;i++){
     for(int j=0;j<key;j++){
            cout<< matrix[i][j]<<" ";
         }
     cout<<"\n";
 }

 string temp2 =keyword;
 sort(temp2.begin(),temp2.end());

 int arr[keyword.size()];
 int check[keyword.size()]={};
 int szk=keyword.size();

 for(int i=0;i<szk;i++){
     for(int j=0;j<szk;j++){
         if(temp2[i]==keyword[j] && check[j]!=-1){
             arr[j]=i;
             check[j]=-1;
             break;
         }
     }
 }
```

```
    for(int i=0;i<szk;i++){
       cout<<arr[i]<<" ";
    }
    cout<<"\n";


    string decrypted="";
    for(int i=0;i<row;i++){
        for(int j=0;j<szk;j++){
            cout<<matrix[i][arr[j]]<<" ";
            decrypted.push_back(matrix[i][arr[j]]);
        }
        cout<<"\n";
    }


     cout<<"\n****  decrypted message is: ****\n\n";
     cout<<"\n"<<decrypted<<"\n";



     ofstream fout2;
     fout2.open("doutput.txt");
     fout2<<decrypted;


     fout2.close();
     fin.close();
     fout1.close();



    return 0;
}
```

➢ **INPUT:**

INPUT FILE (PLAIN TEXT):

input.txt

```
1   CORNELL
2   THEQUICKBROWNFOXJUMPEDOVERTHELAZYDOG
3
```

➢ **OUTPUT:**

```
key word:CORNELL
7
CORNELL
CELLNOR
C 0
E 1
L 2
L 3
N 4
O 5
R 6

**** plain text is ****

THEQUICKBROWNFOXJUMPEDOVERTHELAZYDOG

**** plain text matrix is ****

T H E Q U I C
K B R O W N F
O X J U M P E
D O V E R T H
E L A Z Y D O
G
```

```
****  encrypted message is: ****

TKODEGUWMRY INPTD CFEHO QOUEZ HBXOL ERJVA

T U I C Q H E
K W N F O B R
O M P E U X J
D R T H E O V
E Y D O Z L A
G

0 5 6 4 1 2 3
T H E Q U I C
K B R O W N F
O X J U M P E
D O V E R T H
E L A Z Y D O
G


****  decrypted message is: ****


THEQUICKBROWNFOXJUMPEDOVERTHELAZYDOG
```

OUTPUT FILE (ENCRYPTED TEXT):

output.txt

```
1   TKODEGUWMRY INPTD CFEHO QOUEZ HBXOL ERJVA
2
3
```

DOUTPUT FILE (DECRYPTED TEXT):

doutput.txt

```
1   THEQUICKBROWNFOXJUMPEDOVERTHELAZYDOG
2
3
```

3.3 One time pad

- The message is represented as a binary string (a sequence of 0's and 1's using a coding mechanism such as ASCII coding.
- The key is a truly random sequence of 0's and 1's of the same length as the message.
- message ='IF'
- then its ASCII code =(1001001 1000110)
- key = (1010110 0110001)
- *Encryption:*
- 1001001 1000110 plaintext
- 1010110 0110001 key
- 0011111 1110110 ciphertext

➢ METHODOLOGY FOLLOWED:

```
#include <iostream>
#include<bits/stdc++.h>

using namespace std;

string binary_string(char ch){
   int a=ch;
   string ans="";
   while(a!=0){
      ans.push_back((a%2==0)?'0': '1');
      a/=2;
   }
   int sz=ans.size();

   for(int i=sz;i<8;i++){
    ans.push_back('0');
   }
   reverse(ans.begin(),ans.end());

   return ans;
}

int main()
```

```
{
   // encryption

   ifstream fin;
   fin.open("input.txt");

   ofstream fout;
   fout.open("output.txt");

    string st;

    char ch;


    while(fin.get(ch)){
      st.append(binary_string(ch));
    }

    string key="";
    int n =st.length();

    for(int i=0 ; i<n ; i++ ){
        key.push_back(48+rand()%2);
    }
    cout<<key<<"\n";

    string encryption="";

    for(int i=0;i<n;i++){
        if(st[i]==key[i]){
            encryption.push_back('0');
        }
        else{
            encryption.push_back('1');
        }
    }

    cout<<encryption<<"\n";

    fout<<encryption<<"\n";
    fout.close();
    fin.close();


    //decryption
   char ch1;
   ifstream fin2;
   fin2.open("output.txt");
```

```
    ofstream fout2;
    fout2.open("doutput.txt");

    string decryption_binary="";
    int i=0;
    while(fin2.get(ch1)){
       if( ch1==key[i] ){
         decryption_binary.push_back('0');
       }
       else{
         decryption_binary.push_back('1');
       }
       i++;
    }
    string decryption="";

    for(int j=0;j<n;j+=8){

          int nch=0;
          int pow2 =1;
          for(int l=j+7;l>=j;l--){
             if(decryption_binary[l]=='1'){
                 nch+=pow2;
             }
             pow2*=2;
          }

          decryption.push_back(nch);
    }

    fout2<<decryption<<"\n";
    cout<<decryption<<"\n";

     fout2.close();
     fin2.close();

     return 0;
}
```

➢ INPUT:

INPUT FILE (PLAIN TEXT):

input.txt

```
1    abcdefgh12345
2
3
```

➢ OUTPUT:

```
plain text in form of binary:
01100001011000100110001101100100011001010110011001100111011010000011000100110010001100110011010000110101
generated key:
11001000001111111010100100100110101011101101110100111111001000000000101000110110000000100101100011111 0
encrypted text:
10101001010111011100101001000010110010111011110111000000100011000011000001110100111100111010001000001011
decrypted text:
abcdefgh12345
```

OUTPUT FILE (ENCRYPTED TEXT):

output.txt

```
1  10101001010111011100101001000010110010111011110111000000100011000011000001110100111100111010001000001011
2
3
4
```

DOUTPUT FILE (DECRYPTED TEXT):

doutput.txt

```
1    abcdefgh12345
2
3
4
```