Forth stage: 1st sem.

Columnar Transposition Cipher

Given a plain-text message and a numeric key, cipher/de-cipher the given text using Columnar Transposition Cipher

The Columnar Transposition Cipher is a form of transposition cipher. Columnar Transposition involves writing the plaintext out in rows, and then reading the ciphertext off in columns one by one.

Encryption

In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text.

1. The message is written out in rows of a fixed length, and then read out again column by column, and the columns are chosen in some scrambled order.
2. Width of the rows and the permutation of the columns are usually defined by a keyword.
3. For example, the word HACK is of length 4 (so the column are of length 4), and the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order would be “3 1 2 4”.
4. Any spare spaces are filled with nulls or left blank or placed by a character (Example: -).

5.Finally, the message is read off in columns, in the order specified by the keyword.

Encryption

Given text = Geeks for Geeks

Keyword = HACK **Length of Keyword** = 4 (no of rows) **Order of Alphabets in HACK** = 3124

H	A	C	K
3	1	2	4
G	e	e	k
s	_	f	o
r	_	G	e
e	k	s	_

Print Characters of column 1,2,3,4

Encrypted Text = e kefGsGsrekoe_

Decryption

1. To decipher it, the recipient has to work out the column lengths by dividing the message length by the key length.
2. Then, write the message out in columns again, then re-order the columns by reforming the key word.

Example:

The key for the transposition cipher is a keyword e.g. '**pangram**'. To encrypt a piece of text, e.g. '**The quick brown fox jumps over the lazy dog**', we write it out in a special way in a number of rows:

p	**a**	**n**	**g**	**r**	**a**	**m**
6	**1**	**5**	**3**	**7**	**2**	**4**
T	h	e	-	q	u	i
c	k	-	b	r	o	w
n	-	f	o	x	-	j
u	m	p	s	-	o	v
e	r	-	t	h	e	-
l	a	z	y	-	d	o
g	-	-	-	-	-	-

The ciphertext is read off along the columns: '**hk-mra-uo-oed-bosty-iwjv-o-e-fp-z-Tcnuelgqrx- h- -**'

Decrypt the above encrypted massage

```
// C++ program for illustrating

// Columnar Transposition Cipher

#include<iostream>

#include<string>

using namespace std;   // In programming, we cannot have variables, functions, etc with the same name. So to avoid those conflicts we use  namespaces.

  // Key for Columnar Transposition

string const key = "HACK"; // sorted associative container that contains key

map<int, int> keyMap; // map is a way to store a key-value pair; create an empty map, then add elements to it

  void setPermutationOrder()

{

    // Add the permutation order into map  A permutation also called an

“arrangement number” or “order,” is a rearrangement of the elements of an ordered list


    for(int i=0; i < key.length(); i++)

    {

        keyMap[key[i]] = i;
```

```
    }
}
```

// **Encryption**

```
string encryptMessage(string msg)
{
    int row,col,j;
    string cipher = "";
        /* calculate column of the matrix*/
    col = key.length();
        /* calculate Maximum row of the matrix*/
    row = msg.length()/col;
        if (msg.length() % col)
       row += 1;
     char matrix[row][col];
     for (int i=0,k=0; i < row; i++) // loop to enter the array elements
    {
       for (int j=0; j<col; )
```

```
    {
       if(msg[k] == '\0')
       {


 /* Adding the padding character '_' */
          matrix[i][j] = '_';
          j++;
       }
                if( isalpha(msg[k]) || msg[k]==' ') //  isalpha() is only true if all characters in the string are letters: Return true if all characters in the string are alphabetic
       {
          /* Adding only space and alphabet into matrix*/
          matrix[i][j] = msg[k];
          j++;
       }
       k++;
    }
  }
```

// iteratorrefers to the position where the key is present in the map; **end()** function is used to return an iterator pointing to past the last element of the map container.

```
    for (map<int,int>::iterator ii = keyMap.begin(); ii!=keyMap.end(); ++ii)
   {
      j=ii->second;
       // getting cipher text from matrix column wise using permuted key
      for (int i=0; i<row; i++)
     {
         if( isalpha(matrix[i][j]) || matrix[i][j]==' ' || matrix[i][j]=='_')
            cipher += matrix[i][j];
      }
   }
    return cipher;
}
```

// Decryption

```
string decryptMessage(string cipher)
{
```

```
/* calculate row and column for cipher Matrix */
int col = key.length();
 int row = cipher.length()/col;
char cipherMat[row][col];
 /* add character into matrix column wise */
for (int j=0,k=0; j<col; j++)
   for (int i=0; i<row; i++)
       cipherMat[i][j] = cipher[k++];
 /* update the order of key for decryption */
int index = 0;
for( map<int,int>::iterator ii=keyMap.begin(); ii!=keyMap.end(); ++ii)
   ii->second = index++;
 /* Arrange the matrix column wise according
to permutation order by adding into new matrix */
char decCipher[row][col];
map<int,int>::iterator ii=keyMap.begin();
int k = 0;
for (int l=0,j; key[l]!='\0'; k++)
```

```
{
    j = keyMap[key[l++]];
    for (int i=0; i<row; i++)
    {
        decCipher[i][k]=cipherMat[i][j];
    }
}
```

/* getting Message using matrix */

```
string msg = "";
for (int i=0; i<row; i++)
{
    for(int j=0; j<col; j++)
    {
        if(decCipher[i][j] != '_')
            msg += decCipher[i][j];
    }
}
return msg;
```

```
}

 // Driver Program

int main(void)

{

   /* message */

   string msg = "Geeks for Geeks";

     setPermutationOrder();

        // Calling encryption function

   string cipher = encryptMessage(msg);

   cout << "Encrypted Message: " << cipher << endl;

        // Calling Decryption function

   cout << "Decrypted Message: " << decryptMessage(cipher) << endl;

    return 0;

}
```

Output:

Encrypted Message: e kefGsGsrekoe_

Decrypted Message: Geeks for Geeks