```
#include<bits/stdc++.h>
using namespace std;
// Key for Columnar Transposition
string const key = "HACK";
map<int,int> keyMap;
void setPermutationOrder()
{
// Add the permutation order into map
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++)
{
for (int j=0; j<col; )
{
if(msg[k] == '\0')
{
/* Adding the padding character '_' */
matrix[i][j] = '_';
j++;
}
if( isalpha(msg[k]) || msg[k]==' ')
{
/* Adding only space and alphabet into matrix*/
matrix[i][j] = msg[k];
j++;
}
k++;
}
}
```

```
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 = "College Life";
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: ogilefCeLl e
Decrypted Message: College Life
```