**Cryptography**

### -Lab report



Submitted By: Deshant Devkota

Submitted To: Suresh Thapal (Lecturer, Cryptography)

Vedas College, Lalitpur, Nepal

**Tribhuvan University**

**Department of Bsc Csit**

Submitted Date: Saturday, June 5, 2021

Submitted Lab Reports:

1. To implement Caesar Cipher
2. To implement Playfair Cipher
3. To implement Monoalphabetic Cipher
4. To implement Vignere Cipher
5. To implement Hill Cipher
6. To implement Rail Fence Cipher
7. To implement Row Transpose Cipher

## LAB 1

Caesar Cipher

**1. Objectives:**

In this lab we were to implement Caesar Cipher and use it to encrypt and decrypt a message.

**2. Introduction:**

Caesar Cipher is a substitution Cipher technique in which each letter of plain text is replaced by a letter with some fixed number of positions down with the alphabet. It is named after Julius Caesar since he introduced the use of this cipher.

Encryption with Caesar code is based on an alphabet shift (move of letters further in the alphabet), it is a monoalphabetic substitution cipher, ie. a same letter is replaced with only one other (always the same for a given cipher message). The most commonly used shift/offset is by 3 letters

**4.Code:**

//Implementing Caesar Cipher in C++

#include<iostream>

#include<string>

#include<algorithm>

using namespace std;

void remove\_LC\_spaces(string& text)

{

string formatted\_text;

for(auto text\_single\_char:text)

{

if (text\_single\_char!=' ')

{

{formatted\_text += tolower(text\_single\_char);}

}

}

text = formatted\_text;

}

string encrypt\_using\_caesar(string text)

{

remove\_LC\_spaces(text);

for(auto& single\_text\_char:text)

{

single\_text\_char = (((single\_text\_char+3-97)%26)+97);

}

return text;

}

string decrypt\_using\_caesar(string text)

{

remove\_LC\_spaces(text);

for(auto& single\_text\_char:text)

{

single\_text\_char = (((single\_text\_char-3-97+26)%26)+97);

}

return text;

}

int main()

{

string plain\_text, cipher\_text, decipher\_text;

cout<<"Enter plain text: ";

getline(cin,plain\_text);

cipher\_text = encrypt\_using\_caesar(plain\_text);

cout<<"\nCipher text is: "<<cipher\_text<<endl;

decipher\_text = decrypt\_using\_caesar(cipher\_text);

cout<<"\nDecipher text is: "<<decipher\_text<<endl;

}

**5.Output:**

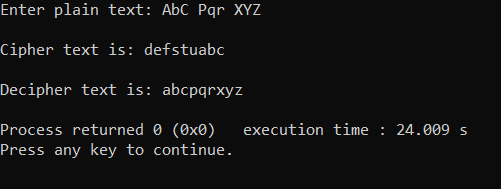


Fig:Output for Encryption and Decryption using Caesar Cipher

## LAB 2

PlayFair Cipher

**1. Objectives:**

In this lab we were to implement Playfair Cipher and use it to encrypt and decrypt a message.

**2. Introduction:**

The Playfair cipher is a manual symmetric encryption technique and was the first literal diagram substitution cipher. The scheme was invented in 1854 by Charles Wheatstone. Lord Playfair is credited for promoting its use.

**4.Code:**

//Implementing Playfair Cipher in C++

#include<iostream>

#include<vector>

#include<map>

#include<string>

#include<utility>

#include <algorithm>

using namespace std;

string remove\_LC\_spaces\_j(const string& text)

{

string return\_text;

for(auto text\_single\_char:text)

{

if (text\_single\_char!=' ')

{

if (text\_single\_char=='j' || text\_single\_char=='J')

{return\_text += 'i';}

else

{return\_text += tolower(text\_single\_char);}

}

}

return return\_text;

}

void display\_playfair\_matrix(char playfair\_matrix[5][5])

{

for(int row=0;row<5;row++)

{

for(int column=0;column<5;column++)

{

cout<<playfair\_matrix[row][column]<<" ";

}

cout<<endl;

}

}

void generate\_playfair\_matrix(char playfair\_matrix[5][5], string key)

{

int row=0, column=0;

map<char, int> mp;

mp['j']=1;

for(auto key\_single\_char:key)

{

if(mp[key\_single\_char]==0)

{

playfair\_matrix[row][column]=key\_single\_char;

mp[key\_single\_char]=1;

column = (column+1)%5;

if (column==0)

row = (row+1)%5;

}

}

for(char alphabet='a';alphabet<='z';alphabet++)

{

if(mp[alphabet]==0)

{

playfair\_matrix[row][column]=alphabet;

mp[alphabet]=1;

column = (column+1)%5;

if (column==0)

row = (row+1)%5;

}

}

}

map<char, pair<int, int>> map\_char\_pos(const char playfair\_matrix[5][5])

{

map<char, pair<int, int>> char\_row\_column\_map;char element;

for(int row=0;row<5;row++)

{

for(int column=0;column<5;column++)

{

element = playfair\_matrix[row][column];

char\_row\_column\_map[element]=make\_pair(row,column);

}

}

return char\_row\_column\_map;

}

vector<pair<char,char>> divide\_string\_to\_pairs(string plain\_text)

{

vector<pair<char,char>> plain\_text\_pair\_vector;

for(int i=0;i<plain\_text.size();i++)

{

if(i!=plain\_text.size()-1)

{

if(plain\_text[i]!=plain\_text[i+1])

{

plain\_text\_pair\_vector.push\_back(make\_pair(plain\_text[i],plain\_text[i+1]));

//plain\_text\_pair\_vector[i].first=plain\_text[i];

//plain\_text\_pair\_vector[i+1].first=plain\_text[i];

i++;

}

else if(plain\_text[i]==plain\_text[i+1])

{

plain\_text\_pair\_vector.push\_back(make\_pair(plain\_text[i],'x'));

}

}

else

{

plain\_text\_pair\_vector.push\_back(make\_pair(plain\_text[i],'x'));

}

}

return plain\_text\_pair\_vector;

}

string encrypt\_using\_play\_fair(string plain\_text, string key)

{

char playfair\_matrix[5][5];

plain\_text=remove\_LC\_spaces\_j(plain\_text);

key=remove\_LC\_spaces\_j(key);

generate\_playfair\_matrix(playfair\_matrix, key);

cout<<"\nThe 5\*5 Play Fair Matrix or Key Table is:\n";

display\_playfair\_matrix(playfair\_matrix);

map<char, pair<int,int>> char\_row\_column\_map = map\_char\_pos(playfair\_matrix);

cout<<"\nThe row and column for each char in the 5\*5 table is: \n";

for(auto iter:char\_row\_column\_map)

{

cout<<iter.first<<":<"<<iter.second.first<<","<<iter.second.second<<">"<<endl;

}

vector<pair<char,char>> plain\_text\_pair\_vector = divide\_string\_to\_pairs(plain\_text);

cout<<"\nThe Plain Text divided into Pairs: \n";

for(auto single\_pair\_item:plain\_text\_pair\_vector)

{

cout<<single\_pair\_item.first<<","<<single\_pair\_item.second<<endl;

}

//The main work

//Generating Cipher Text

//using the pairs which are inserted in vector

string ciphertext;

char first\_letter, second\_letter;

int flr,flc,slr,slc;

for(auto single\_pair\_item:plain\_text\_pair\_vector)

{

first\_letter=single\_pair\_item.first;

second\_letter=single\_pair\_item.second;

flr=char\_row\_column\_map[first\_letter].first;

flc=char\_row\_column\_map[first\_letter].second;

slr=char\_row\_column\_map[second\_letter].first;

slc=char\_row\_column\_map[second\_letter].second;

if(flr==slr)

{

ciphertext+=playfair\_matrix[flr][(flc+1)%5];

ciphertext+=playfair\_matrix[slr][(slc+1)%5];

}

else if(flc==slc)

{

ciphertext+=playfair\_matrix[(flr+1)%5][flc];

ciphertext+=playfair\_matrix[(slr+1)%5][slc];

}

else

{

ciphertext+=playfair\_matrix[flr][slc];

ciphertext+=playfair\_matrix[slr][flc];

}

}

return ciphertext;

}

string decrypt\_using\_play\_fair(string plain\_text, string key)

{

cout<<"\nThis is Decryption Process:\n";

char playfair\_matrix[5][5]{NULL};

plain\_text=remove\_LC\_spaces\_j(plain\_text);

key=remove\_LC\_spaces\_j(key);

generate\_playfair\_matrix(playfair\_matrix, key);

cout<<"\nThe 5\*5 Play Fair Matrix or Key Table is:\n";

display\_playfair\_matrix(playfair\_matrix);

map<char, pair<int,int>> char\_row\_column\_map = map\_char\_pos(playfair\_matrix);

cout<<"\nThe row and column for each char in the 5\*5 table is: \n";

for(auto iter:char\_row\_column\_map)

{

cout<<iter.first<<":<"<<iter.second.first<<","<<iter.second.second<<">"<<endl;

}

vector<pair<char,char>> plain\_text\_pair\_vector = divide\_string\_to\_pairs(plain\_text);

cout<<"\nThe Cipher Text divided into Pairs: \n";

for(auto single\_pair\_item:plain\_text\_pair\_vector)

{

cout<<single\_pair\_item.first<<","<<single\_pair\_item.second<<endl;

}

//The main work Generating Cipher Text using the pairs which are inserted in vector

string ciphertext;char first\_letter, second\_letter; int flr,flc,slr,slc;

for(auto single\_pair\_item:plain\_text\_pair\_vector)

{

first\_letter=single\_pair\_item.first;

second\_letter=single\_pair\_item.second;

flr=char\_row\_column\_map[first\_letter].first;

flc=char\_row\_column\_map[first\_letter].second;

slr=char\_row\_column\_map[second\_letter].first;

slc=char\_row\_column\_map[second\_letter].second;

if(flr==slr)

{

ciphertext+=playfair\_matrix[flr][(flc-1+5)%5];

ciphertext+=playfair\_matrix[slr][(slc-1+5)%5];

}

else if(flc==slc)

{

ciphertext+=playfair\_matrix[(flr-1+5)%5][flc];

ciphertext+=playfair\_matrix[(slr-1+5)%5][slc];

}

else

{

ciphertext+=playfair\_matrix[flr][slc];

ciphertext+=playfair\_matrix[slr][flc];

}

}

return ciphertext;

}

int main()

{

string plain\_text, key, cipher\_text, decipher\_text;

cout<<"Enter plain text: ";

getline(cin,plain\_text);

cout<<"\nEnter Key: ";

getline(cin,key);

cipher\_text = encrypt\_using\_play\_fair(plain\_text,key);

cout<<"\nCipher text is: "<<cipher\_text<<endl;

decipher\_text = decrypt\_using\_play\_fair(cipher\_text,key);

cout<<"\nDecipher text is: "<<decipher\_text<<endl;

}

**5.Output:**

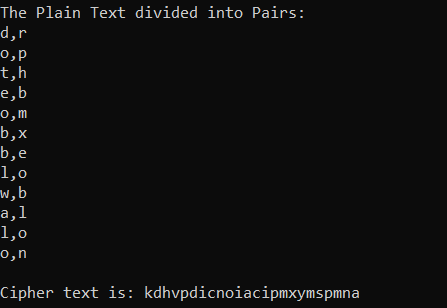
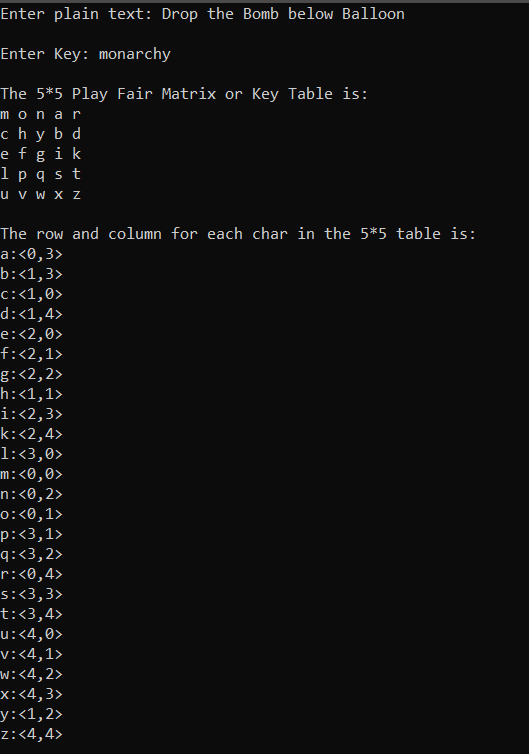


Fig: Output for Encryption using PlayFair Cipher

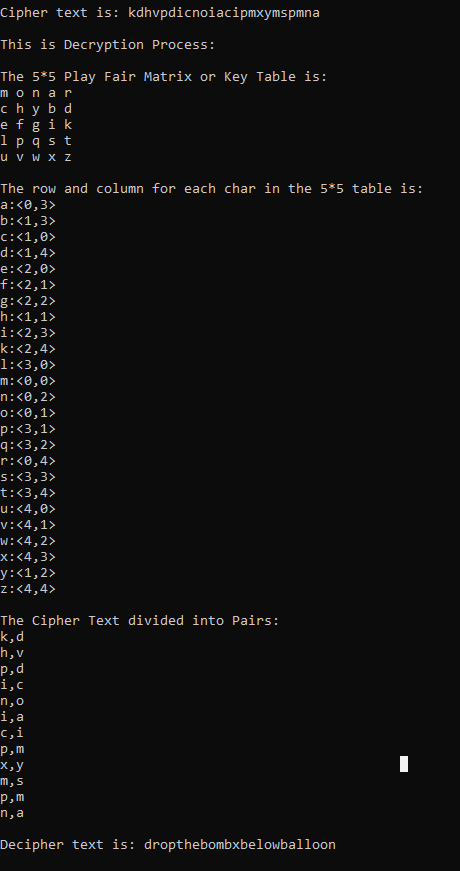


Fig: Output for Decryption using Playfair Cipher

## LAB 3

Mono-alphabetic Cipher

**1. Objectives:**

In this lab we were to implement Mono-Alphabetic Cipher and use it to encrypt and decrypt a message.

**2.Introduction:**

A monoalphabetic substitution cipher, also known as a simple substitution cipher, relies on a fixed replacement structure. That is, the substitution is fixed for each letter of the alphabet. Thus, if "a" is encrypted to "R", then every time we see the letter "a" in the plaintext, we replace it with the letter "R" in the ciphertext.

**3.Code:**

//Code to implement mono-alphabetic cipher using C++

#include<iostream>

#include<map>

#include<string>

#include<algorithm>

using namespace std;

void remove\_UC\_spaces(string& text)

{

string formatted\_text;

for(auto text\_single\_char:text)

{

if (text\_single\_char!=' ')

{

formatted\_text += tolower(text\_single\_char);

}

}

text = formatted\_text;

}

int main()

{

string plain\_text, cipher\_text, decipher\_text;

char letter;

map<char,char> key\_map;

cout<<"Enter key map to map alphabets: \n";

for(char i='a'; i<='z'; i++)

{

cout<<i<<" => ";

cin>>letter;

key\_map[i]=toupper(letter);

}

fflush(stdin);

cout<<"\nEnter plain text: ";

getline(cin,plain\_text);

remove\_UC\_spaces(plain\_text);

for(auto letter:plain\_text)

{

cipher\_text += key\_map[letter];

}

cout<<"\nCipher text is: "<<cipher\_text<<endl;

//Reverse mapping

for(auto letter:cipher\_text)

{

for(auto iter:key\_map)

{

if(iter.second==letter)

decipher\_text+=iter.first;

}

}

cout<<"\nDecipher text is: "<<decipher\_text<<endl;

}

**4.Output:**

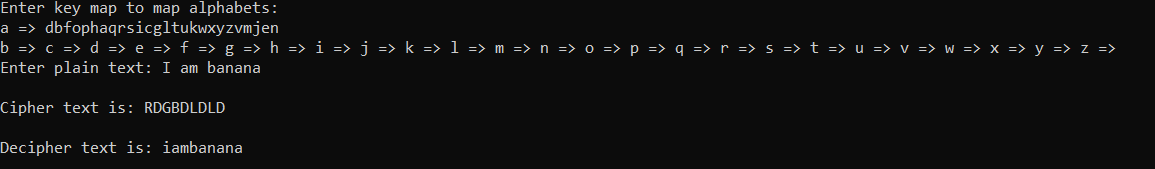


Fig:Output for Encryption and Decryption using Mono-alphabetic Cipher

## LAB 4

Vigenère Cipher

**1.Objective**

In this lab we were to implement Vigenère Cipher and use it to encrypt a message.

**2.Introduction**

Vigenere Cipher is a method of encrypting alphabetic text. It uses a simple form of polyalphabetic substitution. A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabet .

The encryption of the original text is done using the Vigenère square or Vigenère table. The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar Ciphers. At different points in the encryption process, the cipher uses a different alphabet from one of the rows. The alphabet used at each point depends on a repeating keyword.

**3.Code:**

//Code to implement Vigenere Cipher in C++ to encrypt a message

#include<iostream>

#include<string>

#include<algorithm>

using namespace std;

void remove\_LC\_spaces(string& text)

{

string formatted\_text;

for(auto text\_single\_char:text)

{

if (text\_single\_char!=' ')

{

{formatted\_text += tolower(text\_single\_char);}

}

}

text = formatted\_text;

}

void fit\_key\_to\_text(string& key, int ts, int ks)

{

if(ts>ks)

{

for(int i=0;i<ts-ks;i++)

{

key.push\_back(key[i%ks]);

}

}

}

string encrypt\_using\_vignere(string text, string key)

{

string cipher\_text;

remove\_LC\_spaces(text);

remove\_LC\_spaces(key);

fit\_key\_to\_text(key,text.size(),key.size());

cout<<"\nNew key: "<<key<<endl;

for(int i=0;i<text.size();i++)

{

cipher\_text+=((((text[i]-'a')+(key[i]-'a'))%26)+97);

}

return cipher\_text;

}

int main()

{

string plain\_text, key, cipher\_text;

cout<<"Enter plain text: ";

getline(cin,plain\_text);

cout<<"\nEnter key: ";

getline(cin,key);

cipher\_text = encrypt\_using\_vignere(plain\_text,key);

cout<<"\nCipher text is: "<<cipher\_text<<endl;

}

**4.Output:**

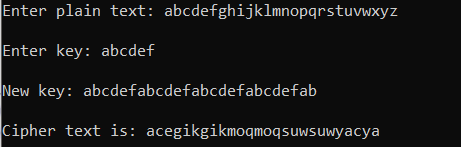


Fig: Encryption using Vigenere Cipher

## LAB 5

Hill Cipher

**1. Objective:**

In this lab we were to implement Hill Cipher and use it to encrypt a message.

**2.Introduction:**

Hill cipher is a polygraphic substitution cipher based on linear algebra.Each letter is represented by a number modulo 26. Often the simple scheme A = 0, B = 1, …, Z = 25 is used, but this is not an essential feature of the cipher. To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.

The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices (modulo 26).

**3.Code:**

//Code to implement Hill Cipher in C++ to encrypt a message

#include<iostream>

#include<string>

#include<math.h>

using namespace std;

void remove\_LC\_spaces(string& text)

{

string formatted\_text;

for(auto text\_single\_char:text)

if (text\_single\_char!=' ')

formatted\_text += tolower(text\_single\_char);

text = formatted\_text;

}

void create\_key\_matrix(char\* key\_matrix, string key, int block\_size)

{

int ks=key.size();

//fitting key size to no of matrix elements

if(block\_size\*block\_size>ks)

for(int i=0; i<block\_size\*block\_size-ks;i++)

key+='x';

for(int row=0;row<block\_size;row++)

for(int col=0;col<block\_size;col++)

key\_matrix[row\*block\_size+col]=key[row\*block\_size+col];

}

string encrypt\_using\_hill\_cipher(string text, string key, int block\_size)

{

string return\_text;

char product=0;

int ts=text.size();

char key\_matrix[block\_size][block\_size];

remove\_LC\_spaces(text);

ts=text.size();

remove\_LC\_spaces(key);

//key\_matrix is a pointer which points to 1D array of integers (i.e first element of 2D array)

//key\_matrix[0] is a pointer which points to first element of 1D array of integers

//same as \*key\_matrix

create\_key\_matrix(key\_matrix[0], key, block\_size);

//displaying key\_matrix formed

cout<<"\nThe key matrix created is: "<<endl;

for(int i = 0; i < block\_size; i++)

{

for(int j = 0; j < block\_size; j++)

printf("%c ", key\_matrix[i][j]);

printf("\n");

}

//padding text with extra letters to make multiple of block\_size

while(ts%block\_size!=0)

{

text.push\_back('x');

ts=text.size();

}

int m,t,p=0;

//multiplying key\_matrix with text

for(int i=0;((i+1)\*block\_size)<=ts;i++)

{

for(int row=0; row<block\_size; row++)

{

for(int col=0; col<block\_size; col++)

{

m=key\_matrix[row][col]-97;

t=text[i\*block\_size+col]-97;

p+=m\*t;

}

p=(p%26)+97;

product = p;

return\_text+=product;

p=0;

}

}

return return\_text;

}

int main()

{

string plain\_text, key, cipher\_text;

int block\_size;

cout<<"Enter plain text: ";

getline(cin,plain\_text);

cout<<"\nEnter Key: ";

getline(cin,key);

cout<<"\nEnter Block Size: ";

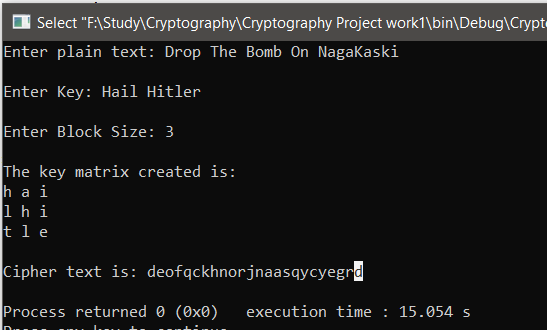
cin>>block\_size;

cipher\_text = encrypt\_using\_hill\_cipher(plain\_text,key,block\_size);

cout<<"\nCipher text is: "<<cipher\_text<<endl;

}

**4.Output:**



## LAB 6

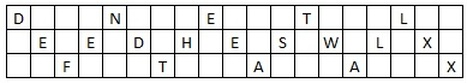
Rail Fence Cipher

**1.Objective:**

In this lab we were to implement Rail fence Cipher and use it to encrypt and decrypt a message.

**2.Introduction:**

The Rail Fence cipher works by writing your message on alternate lines across the page, and then reading off each line in turn. For example, the plaintext "defend the east wall" is written as shown below, with all spaces removed.



**3.Code:**

//

#include<iostream>

#include<string>

#include<algorithm>

using namespace std;

void remove\_LC\_spaces(string& text)

{

string formatted\_text;

for(auto text\_single\_char:text)

if (text\_single\_char!=' ')

formatted\_text += tolower(text\_single\_char);

text = formatted\_text;

}

string encrypt\_using\_rail\_fence(string plain\_text,int key)

{

remove\_LC\_spaces(plain\_text);

int msgLen = plain\_text.size();

char matrix[key][msgLen];

for(int row=0; row<key; row++)

for(int col=0; col<msgLen; col++)

matrix[row][col]='\n';

int row=0; bool dir=0;

for(int col=0; col<msgLen; col++)

{

matrix[row][col]=plain\_text[col];

if(row==0||row==key-1)

dir=!dir;

dir?row++:row--;

}

string cipher\_text;

for(int row=0; row<key; row++)

for(int col=0; col<msgLen; col++)

if(matrix[row][col]!='\n') cipher\_text+=matrix[row][col];

return cipher\_text;

}

string decrypt\_using\_rail\_fence(string text,int key)

{

remove\_LC\_spaces(text);

int msgLen = text.size();

char matrix[key][msgLen];

for(int row=0; row<key; row++)

for(int col=0; col<msgLen; col++)

matrix[row][col]='\n';

int row=0; bool dir=0;

for(int col=0; col<msgLen; col++)

{

matrix[row][col]='\*';

if(row==0||row==key-1)

dir=!dir;

dir?row++:row--;

}

int i=0;

for(int row=0; row<key; row++)

for(int col=0; col<msgLen; col++)

if(matrix[row][col]=='\*') matrix[row][col]=text[i++];

string cipher\_text;

for(int row=0; row<key; row++)

for(int col=0; col<msgLen; col++)

if(matrix[row][col]!='\n') cipher\_text+=matrix[row][col];

return cipher\_text;

}

int main()

{

string plain\_text, cipher\_text, decipher\_text;

int key =3;

cout<<"Enter plain text: ";

getline(cin,plain\_text);

cout<<"\nEnter Key: ";

cin>>key;

cipher\_text = encrypt\_using\_rail\_fence(plain\_text,key);

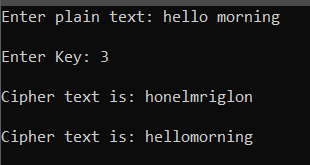
cout<<"\nCipher text is: "<<cipher\_text<<endl;

decipher\_text = decrypt\_using\_rail\_fence(plain\_text,key);

cout<<"\nCipher text is: "<<decipher\_text<<endl;

}

**4.Output:**



## LAB 7

Row Transpose Cipher

**1.Objective:**

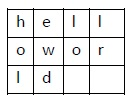
In this lab we were to implement Row Transpose Cipher and use it to encrypt and decrypt a message.

**2.Introduction:**

In row transposition we write down the string in a matrix starting from the first row to the last going through each column. Then we alter the columns in order defined using the key. We then

read the letters from the first column to the last going through each row.

For eg: A string “hello world” is written in a 3\*4 matrix as below



The columns are then exchanged in a order defined by key maybe “3214” and the letters are read from top to bottom fashion which will produce a ciphertext as “lo ewdhollr “.

**3.Code:**

def split\_len(seq, length):

return [seq[i:i + length] for i in range(0, len(seq), length)]

def encode(key, plaintext):

order = {

int(val): num for num, val in enumerate(key)

}

ciphertext = ''

for index in sorted(order.keys()):

for part in split\_len(plaintext, len(key)):

try:

ciphertext += part[order[index]]

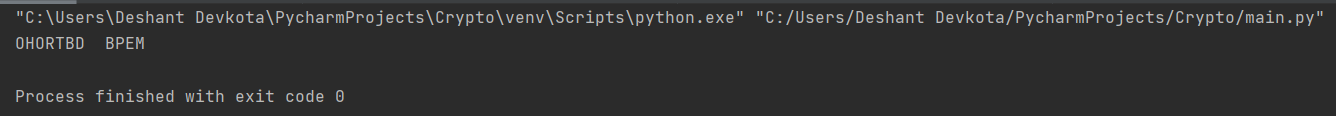
except IndexError:

continue

return ciphertext

print(encode('3214', 'DROP THE BOMB'))

**4.Output:**



============================End===================================