**Final Year B. Tech., Sem VI 2022-23**

**cryptography and network security Lab**

**PRN No: 2019BTECS00071**

**Name: Charpale Abhishek Parmanand**

**Batch: B3**

**Assignment 5 : RailFence and columnar Encryption and Decryption**

**Aim:**

To develop and implement the Transposition Cipher (Rail Fence Cipher And Columnar Cipher) and to encryption and decryption on the input plaintext

**Theory:**

**Rail Fence Cipher:**

* In the rail fence cipher, the plain-text is written downwards and diagonally on successive rails of an imaginary fence.
* When we reach the bottom rail, we traverse upwards moving diagonally, after reaching the top rail, the direction is changed again. Thus the alphabets of the message are written in a zig-zag manner.
* After each alphabet has been written, the individual rows are combined to obtain the cipher-text.

**Columnar Cipher:**

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

* 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.
* Width of the rows and the permutation of the columns are usually defined by a keyword.
* For example, the word HACK is of length 4 (so the rows 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”.
* Any spare spaces are filled with nulls or left blank or placed by a character (Example: \_).
* Finally, the message is read off in columns, in the order specified by the keyword

**Rail Fence:**

**Code :**

#include<bits/stdc++.h>

using namespace std;

string getText(string text)

{

string x="";

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

{

if(text[i] >='a' && text[i]<='z')

x += (text[i]-'a')+'A';

else

x += text[i];

}

return x;

}

string encryption(string text,int key)

{

char rail[key][(text.length())];

text = getText(text);

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

{

for(int j=0;j<text.length();j++)

rail[i][j] = '^';

}

bool dir\_down = false;

int row=0,col=0;

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

{

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

dir\_down = !dir\_down;

rail[row][col++] = text[i];

dir\_down ? row++ : row--;

}

string result;

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

{

for(int j=0;j<text.length();j++)

{

if(rail[i][j] != '^')

result.push\_back(rail[i][j]);

}

}

return result;

}

string decryption(string text,int key)

{

char rail[key][(text.length())];

text = getText(text);

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

{

for(int j=0;j<text.length();j++)

rail[i][j] = '\n';

}

bool dir\_down;

int row=0,col=0;

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

{

if(row==0)

dir\_down = true;

if(row == key-1)

dir\_down = false;

rail[row][col++] = '\*';

dir\_down ? row++ : row--;

}

int index=0;

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

{

for(int j=0;j<text.length();j++)

{

if(rail[i][j] == '\*' && index<text.length())

rail[i][j] = text[index++];

}

}

string result;

row = 0;

col = 0;

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

{

if(row == 0)

dir\_down = true;

if(row == key-1)

dir\_down = false;

if(rail[row][col] != '\*')

result.push\_back(rail[row][col++]);

dir\_down ? row++ : row--;

}

return result;

}

int main()

{

int choice;

int datachoice;

string sample;

int key;

while(1)

{

cout << "Rail Fence Cipher\n 1. Encryption \n 2. Decryption\n 3. Exit\nEnter Choice: ";

cin>>choice;

if(choice>2 || choice <=0)

break;

switch(choice)

{

case 1:

cout<<"Enter data to be Encrypted:\n";

cin.ignore();

getline(cin,sample);

cout<<"Enter the key: ";

cin>>key;

cout<<"Encrypted String:\n";

cout<<encryption(sample,key)<<endl;

break;

case 2:

cout<<"Enter data to be Decrypted:\n";

cin.ignore();

getline(cin,sample);

cout<<"Enter the key: ";

cin>>key;

cout<<"Decrypted String:\n";

cout<<decryption(sample,key)<<endl;;

break;

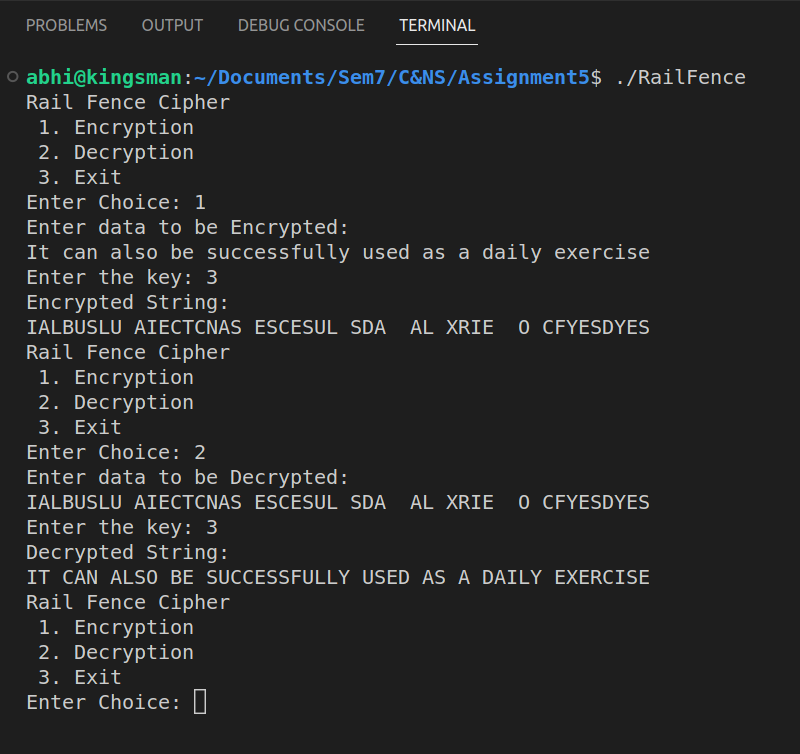
}

}

return 0;

}

**Output:**

****

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Columnar:**

**Code:**

#include<bits/stdc++.h>

using namespace std;

map<int,int> keyMap;

string getText(string text)

{

string x="";

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

{

if(text[i] >='a' && text[i]<='z')

x += (text[i]-'a')+'A';

else

x += text[i];

}

return x;

}

void setPermutationOrder(string key)

{

keyMap.clear();

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

keyMap[key[i]] = i;

}

string encryption(string msg,string key)

{

key = getText(key);

// cout<<key<<endl;

setPermutationOrder(key);

msg = getText(msg);

// cout<<msg<<endl;

int row,col,j;

string cipher="";

col = key.length();

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')

{

matrix[i][j] = '\_';

j++;

}

if(isalpha(msg[k]) || msg[k] == ' ')

{

matrix[i][j] = msg[k];

j++;

}

k++;

}

}

for(map<int,int>::iterator ii = keyMap.begin(); ii != keyMap.end(); ++ii)

{

j = ii->second;

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

{

if(isalpha(matrix[i][j]) || matrix[i][j] == ' ' || matrix[i][j] == '\_')

cipher += matrix[i][j];

}

}

return cipher;

}

string decryption(string cipher,string key)

{

key = getText(key);

// cout<<key<<endl;

setPermutationOrder(key);

cipher = getText(cipher);

// cout<<cipher<<endl;

int col = key.length();

int row = cipher.length()/col;

char cipherMatrix[row][col];

for(int j=0,k=0;j<col;j++)

{

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

cipherMatrix[i][j] = cipher[k++];

}

int index=0;

for(map<int,int>::iterator ii = keyMap.begin(); ii != keyMap.end(); ++ii)

{

ii->second = index++;

}

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] = cipherMatrix[i][j];

}

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;

}

int main()

{

int choice;

int datachoice;

string sample,key;

int shift;

while(1)

{

cout << "Columnar Cipher\n 1. Encryption \n 2. Decryption\n 3. Exit\nEnter Choice: ";

cin>>choice;

if(choice>2)

break;

switch(choice)

{

case 1:

cout<<"Enter data to be Encrypted:\n";

cin.ignore();

getline(cin,sample);

cout<<"Enter the key: ";

getline(cin,key);

cout<<"Encrypted String:\n";

cout<<encryption(sample,key)<<endl;

break;

case 2:

cout<<"Enter data to be Decrypted:\n";

cin.ignore();

getline(cin,sample);

cout<<"Enter the key: ";

getline(cin,key);

cout<<"Decrypted String:\n";

cout<<decryption(sample,key)<<endl;;

break;

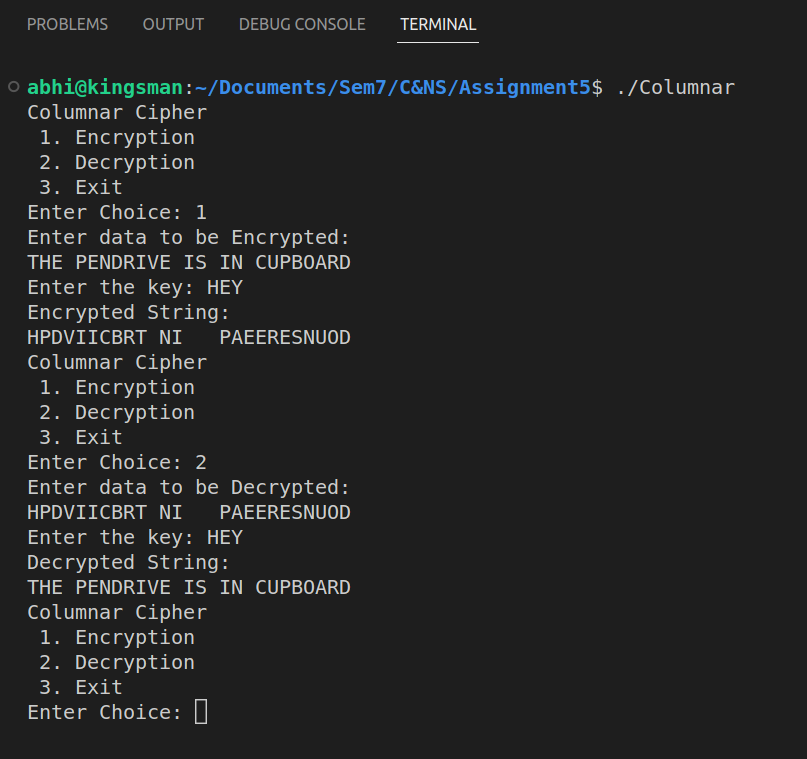
}

}

return 0;

}

**Output:**

****

**Conclusion:**

Performed the experiment successfully. Encrypted the data with the provided key. Output of this encryption is decrypted to match the plaintext that was inputted by the user as shown in the above output.