# **Vernam Cipher**

## **Source Code**

#include<stdio.h>

#include<conio.h>

#include<string.h>

int findindexofalphabet(char alphabet);

void finddecimals(char text[], int textdecimal[20]);

void vernamencryption(char plaintext[], char key[]);

void vernamdecryption(char encrypttext[], char key[]);

int main()

{

char plaintext[20], key[10];

puts("Enter the plaintext:");

gets(plaintext);

puts("Enter the key:");

gets(key);

vernamencryption(plaintext, key);

}

void vernamencryption(char plaintext[], char key[])

{

int plaintextdecimal[20], keydecimal[10], i;

char encrypttext[20], alpha[27]= "abcdefghijklmnopqrstuvwxyz";

if (strlen(plaintext) < strlen(key)){

i=strlen(plaintext);

while (i!=strlen(key)){

plaintext[i]='x';

i++;

}

plaintext[i]='\0';

}

//puts(plaintext);

finddecimals(plaintext, plaintextdecimal);

finddecimals(key, keydecimal);

for (i=0; plaintext[i]!='\0'; i++){ //encrypts the text

encrypttext[i]=alpha[(plaintextdecimal[i]^keydecimal[(i%strlen(key))])%26];

}

encrypttext[i]='\0';

puts("Encrypted Text:");

puts(strupr(encrypttext));

vernamdecryption(strlwr(encrypttext), key);

}

void vernamdecryption(char encrypttext[], char key[])

{

int encryptdecimal[20], keydecimal[10], i;

char decrypttext[20], alpha[27]= "abcdefghijklmnopqrstuvwxyz";

finddecimals(encrypttext, encryptdecimal);

finddecimals(key, keydecimal);

for (i=0; encrypttext[i]!='\0'; i++){ //decrypts the text

decrypttext[i]=alpha[(encryptdecimal[i]^keydecimal[(i%strlen(key))])%26];

}

decrypttext[i]='\0';

puts("Decrypted Text:");

puts(strupr(decrypttext));

}

void finddecimals(char text[], int textdecimal[20]) //finds decimal value of text and stores it in textdecimals

{

int i;

for (i=0; text[i]!='\0'; i++){

textdecimal[i]=findindexofalphabet(text[i]);

//printf("%d\n", textdecimal[i]);

}

}

int findindexofalphabet(char alphabet)

{

char alpha[27]= "abcdefghijklmnopqrstuvwxyz";

int i;

for (i=0; i<26;i++){

if (alphabet==alpha[i]){

return i;

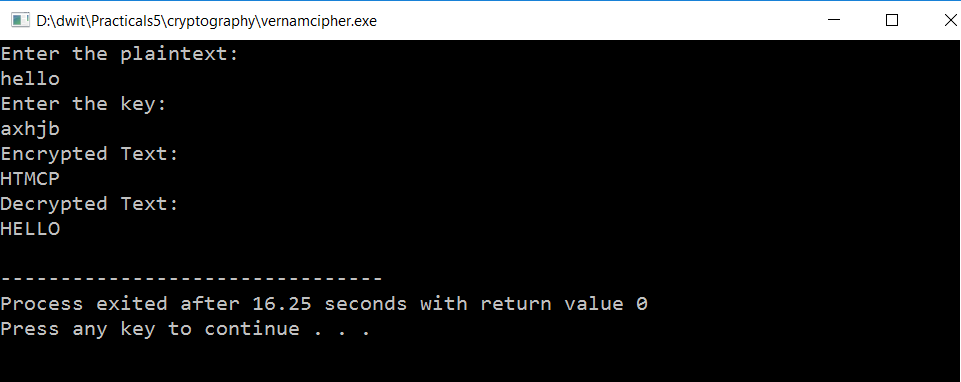
break;

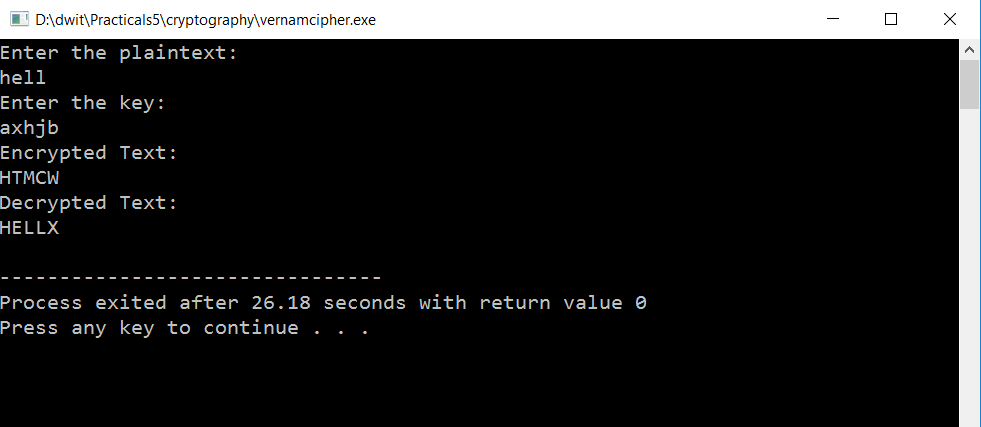
}

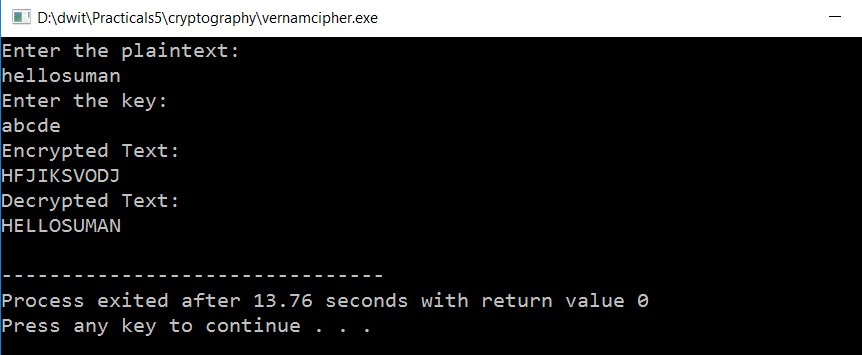
}

}

## **Output**







# **Rail-fence Cipher**

## **Source Code**

#include<stdio.h>

#include<conio.h>

#include<string.h>

void railfenceencryption(char plaintext[], int numberofrails);

void railfencedecryption(char encrypttext[], int numberofrails);

void diagonalpatterngenerator(int diagonalpattern[], int numberofrails);

int main()

{

char plaintext[20];

int numberofrails;

puts("Enter the plaintext:");

gets(plaintext);

puts("Enter the number of rails:");

scanf("%d", &numberofrails);

railfenceencryption(plaintext, numberofrails);

}

void railfenceencryption(char plaintext[], int numberofrails)

{

int limit=numberofrails, length=strlen(plaintext), i, j, diagonalpattern[50], index=0;

char encrypttext[50],matrix[10][50];

while (1){

if (strlen(plaintext)>limit){

limit=limit+2\*(numberofrails-1);

}else{

break;

}

}

while (length!=limit){ //balances the length of plaintext according to numberofrails

plaintext[length]='x';

length++;

}

plaintext[length]='\0';

for (i=0; i<numberofrails;i++){

for (j=0; j<strlen(plaintext); j++){

matrix[i][j]=' ';

}

}

diagonalpatterngenerator(diagonalpattern, numberofrails);

for (j=0; j<strlen(plaintext); j++){ //creates the matrix

matrix[diagonalpattern[j]][j]=plaintext[j];

}

for (i=0; i<numberofrails;i++){

for (j=0; j<strlen(plaintext); j++){

printf("%c ", matrix[i][j] );

}

printf("\n");

}

for (i=0; i<numberofrails;i++){

for (j=0; j<strlen(plaintext); j++){

if (matrix[i][j]!=' '){

encrypttext[index]=matrix[i][j];

index++;

}

}

}

encrypttext[index]='\0';

puts("Encrypted Text:");

puts(strupr(encrypttext));

strlwr(encrypttext);

railfencedecryption(strlwr(encrypttext), numberofrails);

}

void railfencedecryption(char encrypt\_text[], int numberofrails)

{

int i, j, k=0, index=0, diagonalpattern[50];

char decrypt\_text[50],matrix[10][50];

for (i=0; i<numberofrails;i++){

for (j=0; j<strlen(encrypt\_text);j++){

matrix[i][j]=' ';

}

}

for (i=0; i<numberofrails;i++){ //inserts encrypted message in matrix

int flag=0;

for (j=i; j<strlen(encrypt\_text);){

matrix[i][j]=encrypt\_text[index];

index++;

if (i==numberofrails-1 || i==0){

j=j+2\*(numberofrails-1);

}else{

if (flag==0){

j=j+2\*(numberofrails-(i+1));

flag=1;

}else{

j=j+k;

flag=0;

}

}

}

k=k+2;

}

for (i=0; i<numberofrails;i++){

for (j=0; j<strlen(encrypt\_text);j++){

printf("%c ", matrix[i][j]);

}

printf("\n");

}

index=0;

diagonalpatterngenerator(diagonalpattern, numberofrails);

for (j=0; j<strlen(encrypt\_text); j++){ //decodes the matrix

decrypt\_text[index++]=matrix[diagonalpattern[j]][j];

}

decrypt\_text[index]='\0';

puts("Decrypted Text:");

puts(strupr(decrypt\_text));

}

void diagonalpatterngenerator(int diagonalpattern[], int numberofrails)

{

int i;

for (i=0; i<numberofrails; i++){

diagonalpattern[i]=i;

}

for (i=numberofrails; i<50;i++){

if (diagonalpattern[i-1]==numberofrails-1){

diagonalpattern[i]=diagonalpattern[i-1]-1;

}else{

if (diagonalpattern[i-1]==0){

diagonalpattern[i]=diagonalpattern[i-1]+1;

}else{

if (diagonalpattern[i-2]==diagonalpattern[i-1]+1){

diagonalpattern[i]=diagonalpattern[i-1]-1;

}

if (diagonalpattern[i-2]==diagonalpattern[i-1]-1){

diagonalpattern[i]=diagonalpattern[i-1]+1;

}

}

}

}

}

## **Output**



