**Practical – 1**

**Aim: To implement Caesar cipher encryption-decryption.**

**Code:**

#include<iostream>

using namespace std;

char cipherAry[20];

void plainTextToCipherTextCeasorCipher(string plainText,int key)

{

int cipher,num;

for(int i=0; plainText[i]!=NULL; i++)

{

if(plainText[i]>=65 && plainText[i]<=90)

{

num = plainText[i]-65;

}

else if(plainText[i]>=97 && plainText[i]<=122)

{

num = plainText[i]-97;

}

cipher =(num + key)%26;

cipherAry[i]=cipher+65;

cout<<cipherAry[i];

}

}

void cipherTextToPlainTextCeasorCipher(int key)

{

int cipher,num;

char plainText[20];

for(int i=0; cipherAry[i]!=NULL; i++)

{

if(cipherAry[i]>=65 && cipherAry[i]<=90)

{

num = cipherAry[i]-65;

}

else if(cipherAry[i]>=97 && cipherAry[i]<=122)

{

num = cipherAry[i]-97;

}

cipher =(num - key)%26;

plainText[i]=cipher+97;

cout<<plainText[i];

}

}

int main()

{

string plainText,cipherText;

int key =0;

cout<<"Enter plaintext : ";

cin>>plainText;

cout<<"Enter key : ";

cin>>key;

cout<<"\n\n-----Plaintext to Ciphertext-----"<<endl<<endl;

cout<<"Plain text : "<<plainText<<endl;

cout<<"Key : "<<key<<endl;

cout<<"Cipher text : ";

plainTextToCipherTextCeasorCipher(plainText,key);

cout<<endl;

cout<<"\n\n-----Ciphertext to Plaintext-----"<<endl<<endl;

cout<<"Cipher text : ";

plainTextToCipherTextCeasorCipher(plainText,key);

cout<<endl;

cout<<"Key : "<<key<<endl;

cout<<"Plain text : ";

cipherTextToPlainTextCeasorCipher(key);

cout<<endl;

return 0;

}

**Output:**

A computer screen shot of a black screen

Description automatically generated

**Conclusion:**

* In the Caesar cipher, each letter of the plaintext is replaced by another letter according to the formula: c = (plaintext + key) mod 26.

**For ex.** d is replace with G

* In the decryption process, each letter is replaced with another letter according to the formula: plaintext = (ciphertext - key).

**For ex.** G is replace with d

**Practical – 2**

**Aim: To implement Monoalphabetic cipher encryption-decryption.**

**Code:**

#include<iostream>

using namespace std;

int main()

{

char plainText[20],cipherText[20],cipher[26],key[25],smpAry[26],decAry[26];

int k=0,p=0;

bool flag;

cout<<"-----Mono-Alphabetic Cipher-----"<<endl<<endl;

cout<<"Enter plaintext : ";

cin>>plainText;

cout<<"Enter key : ";

cin>>key;

for(int i=0; plainText[i]!=NULL; i++)

{

if(plainText[i]>='A' && plainText[i]<='Z')

{

plainText[i]=plainText[i]+32;

}

}

for(int i=0; key[i]!=NULL; i++)

{

if(key[i]>='A' && key[i]<='Z')

{

key[i]=key[i]+32;

}

}

for(int i=0; key[i]!=NULL; i++)

{

flag=false;

for(int j=0; cipher[j]!=NULL; j++)

{

if((cipher[j]+32)==key[i])

{

flag=true;

break;

}

}

if(!flag)

{

cipher[k]=key[i]-32;

k++;

}

}

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

{

smpAry[i]=65+i;

}

cout<<endl<<endl;

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

{

cout<<smpAry[i]<<" ";

}

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

{

flag=false;

for(int j=0; cipher[j]!=NULL; j++)

{

if(cipher[j]==smpAry[i])

{

flag=true;

break;

}

}

if(!flag)

{

cipher[k++]=smpAry[i];

}

}

cipher[k]=NULL;

cout<<endl;

for(int i=0; cipher[i]!=NULL; i++)

{

cout<<cipher[i]<<" ";

}

for(int i=0; plainText[i]!=NULL; i++)

{

for(int j=0; j<26; j++)

{

if((plainText[i]-32)== smpAry[j])

{

cipherText[p++]=cipher[j];

}

}

}

cipherText[p]=NULL;

p=0;

for(int i=0; cipherText[i]!=NULL; i++)

{

for(int j=0; j<26; j++)

{

if(cipherText[i]== cipher[j])

{

decAry[p++]=smpAry[j];

}

}

}

decAry[p]==NULL;

cout<<"\n\n-----Plaintext to Ciphertext-----"<<endl<<endl;

cout<<"Plain text : "<<plainText<<endl;

cout<<"Key : "<<key<<endl;

cout<<"Cipher text : "<<cipherText<<endl;;

cout<<endl;

cout<<"\n\n-----Ciphertext to Plaintext-----"<<endl<<endl;

cout<<"Cipher text : "<<cipherText<<endl;;

cout<<"Key : "<<key<<endl;

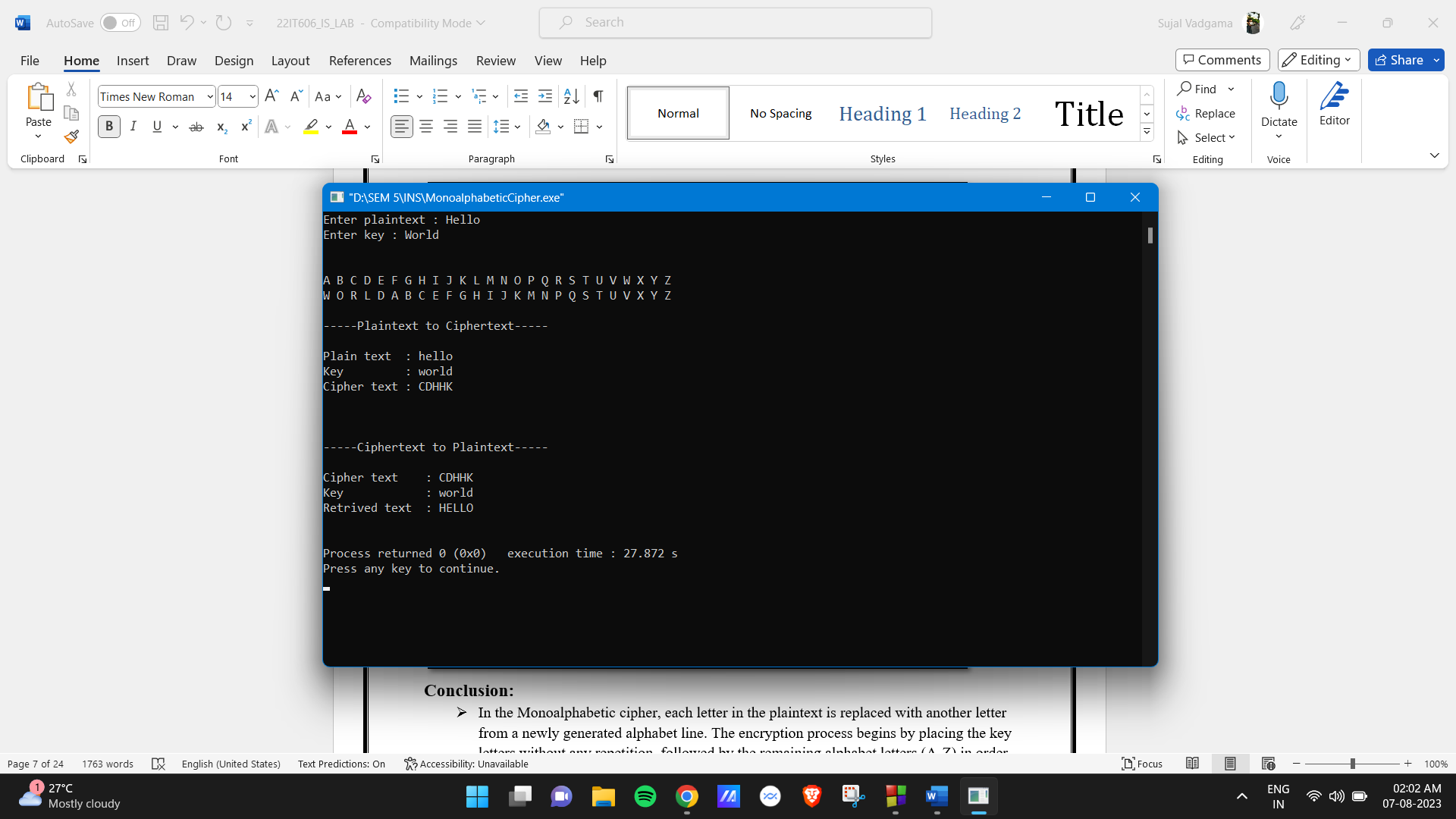
cout<<"Retrived text : "<<decAry<<endl;

cout<<endl;

return 0;

}

**Output:**



**Conclusion:**

* In the Monoalphabetic cipher, each letter in the plaintext is replaced with another letter from a newly generated alphabet line. The encryption process begins by placing the key letters without any repetition, followed by the remaining alphabet letters (A-Z) in order. The plaintext letters are then substituted according to their index positions in the newly formed alphabet line.

**For ex.** h is replaced with F

* And in decryption each letter again replaces with another letter as per shown new line then ciphertext is replaced with alphabets A-Z.

**For ex**. F is replace with h

**Practical – 3**

**Aim: To implement Playfair cipher encryption-decryption.**

**Code:**

#include<iostream>

using namespace std;

int main()

{

char plainText[30],matrix[5][5],key[10],tempKey[25],tempMatrix[25],cipher[25],decAry[25],temp[25];

int k=0,len=0,plen=0,part=0,index1Row,index1Column,index2Row,index2Column,cIndex=0;

int indexAry[30];

bool flag,flag1;;

cout<<"-------------PlayFair Cipher------------"<<endl<<endl;

cout<<"Enter plaintext : ";

cin>>plainText;

cout<<"Enter key : ";

cin>>key;

for(int i=0; plainText[i]!=NULL; i++)

{

if(plainText[i]>='A' && plainText[i]<='Z')

{

plainText[i]=plainText[i]+32;

}

temp[i]=plainText[i];

if(plainText[i]=='j' || plainText[i]=='J')

{

plainText[i]='i';

}

plen++;

if(i%2==0)

{

part++;

}

}

for(int i=0; key[i]!=NULL; i++)

{

if(key[i]>='A' && key[i]<='Z')

{

key[i]=key[i]+32;

}

if(key[i]=='j' || key[i]=='J')

{

key[i]='i';

}

}

for(int i=0; key[i]!=NULL; i++)

{

flag=false;

for(int j=0; tempKey[j]!=NULL; j++)

{

if((tempKey[j]+32)==key[i])

{

flag=true;

break;

}

}

if(!flag)

{

tempKey[k]=key[i]-32;

k++;

}

}

len=k;

for(char i='A'; i<='Z'; i++)

{

if(i=='J')

{

i++;

}

flag=false;

for(int j=0; tempKey[j]!=NULL; j++)

{

if(i==tempKey[j])

{

flag=true;

break;

}

}

if(!flag)

{

tempKey[k]=i;

k++;

}

}

len=0;

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

{

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

{

matrix[i][j]=tempKey[len++];

}

}

cout<<"\n\n-----Matrix-----"<<endl<<endl;

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

{

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

{

cout<<matrix[i][j]<<" ";

}

cout<<endl;

}

cout<<"\n\n----------Plaintext divide in 2 char ---------"<<endl<<endl;

k=0;

for(int i=0; plainText[i]!=NULL; i++)

{

if(plainText[i]==plainText[i+1])

{

plen++;

for(int j=plen; j >= i+1; j--)

{

plainText[j]=plainText[j-1];

}

if(plainText[i] == 'x')

{

plainText[i+1]='y';

}

else

{

plainText[i+1]='x';

}

indexAry[k]=i;

k++;

}

i++;

}

if(plen % 2 != 0)

{

if(plainText[plen - 1] == 'x')

{

plainText[plen]='y';

}

else

{

plainText[plen]='x';

}

plen++;

}

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

{

if(i%2==0 && i!=0)

{

cout<<" ";

}

plainText[i]-=32;

cout<<plainText[i];

}

for(int i=0; plainText[i]!=NULL; i++)

{

flag=flag1=false;

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

{

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

{

if(plainText[i] == matrix[j][l])

{

index1Row=j;

index1Column=l;

flag=true;

}

if(plainText[i+1] == matrix[j][l])

{

index2Row=j;

index2Column=l;

flag1=true;

}

if(flag && flag1)

{

break;

}

}

if(flag && flag1)

{

break;

}

}

if(index1Row==index2Row)

{

if(index1Column==4)

{

cipher[cIndex]=matrix[index1Row][0];

cIndex++;

cipher[cIndex]=matrix[index2Row][index2Column+1];

cIndex++;

}

else if(index2Column==4)

{

cipher[cIndex]=matrix[index1Row][index1Column+1];

cIndex++;

cipher[cIndex]=matrix[index2Row][0];

cIndex++;

}

else

{

cipher[cIndex]=matrix[index1Row][index1Column+1];

cIndex++;

cipher[cIndex]=matrix[index2Row][index2Column+1];

cIndex++;

}

}

else if(index1Column==index2Column)

{

if(index1Row==4)

{

cipher[cIndex]=matrix[0][index1Column];

cIndex++;

cipher[cIndex]=matrix[index2Row+1][index2Column];

cIndex++;

}

else if(index2Row==4)

{

cipher[cIndex]=matrix[index1Row+1][index1Column];

cIndex++;

cipher[cIndex]=matrix[0][index2Column];

cIndex++;

}

else

{

cipher[cIndex]=matrix[index1Row+1][index1Column];

cIndex++;

cipher[cIndex]=matrix[index2Row+1][index2Column];

cIndex++;

}

}

else

{

cipher[cIndex]=matrix[index1Row][index2Column];

cIndex++;

cipher[cIndex]=matrix[index2Row][index1Column];

cIndex++;

}

flag=flag1=false;

i++;

}

cout<<"\n\n-----Plaintext to Ciphertext-----"<<endl<<endl;

cout<<"Plain text : ";

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

{

cout<<temp[i];

}

cout<<endl;

cout<<"Key : "<<key<<endl;

cout<<"Cipher text : ";

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

{

cout<<cipher[i];

}

cout<<endl;

cIndex=0;

cout<<"\n\n-----Ciphertext to Plaintext-----"<<endl<<endl;

cout<<"Cipher text : ";

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

{

cout<<cipher[i];

}

cout<<endl;

cout<<"Key : "<<key<<endl;

for(int i=0; cipher[i]!=NULL; i++)

{

flag=flag1=false;

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

{

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

{

if(cipher[i] == matrix[j][l])

{

index1Row=j;

index1Column=l;

flag=true;

}

if(cipher[i+1] == matrix[j][l])

{

index2Row=j;

index2Column=l;

flag1=true;

}

if(flag && flag1)

{

break;

}

}

if(flag && flag1)

{

break;

}

}

if(index1Row==index2Row)

{

if(index1Column==0)

{

decAry[cIndex]=matrix[index1Row][4];

cIndex++;

decAry[cIndex]=matrix[index2Row][index2Column-1];

cIndex++;

}

else if(index2Column==0)

{

decAry[cIndex]=matrix[index1Row][index1Column-1];

cIndex++;

decAry[cIndex]=matrix[index2Row][4];

cIndex++;

}

else

{

decAry[cIndex]=matrix[index1Row][index1Column-1];

cIndex++;

decAry[cIndex]=matrix[index2Row][index2Column-1];

cIndex++;

}

}

else if(index1Column==index2Column)

{

if(index1Row==0)

{

decAry[cIndex]=matrix[4][index1Column];

cIndex++;

decAry[cIndex]=matrix[index2Row-1][index2Column];

cIndex++;

}

else if(index2Row==0)

{

decAry[cIndex]=matrix[index1Row-1][index1Column];

cIndex++;

decAry[cIndex]=matrix[4][index2Column];

cIndex++;

}

else

{

decAry[cIndex]=matrix[index1Row-1][index1Column];

cIndex++;

decAry[cIndex]=matrix[index2Row-1][index2Column];

cIndex++;

}

}

else

{

decAry[cIndex]=matrix[index1Row][index2Column];

cIndex++;

decAry[cIndex]=matrix[index2Row][index1Column];

cIndex++;

}

flag=flag1=false;

i++;

}

cout<<"Retrived text : ";

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

{

decAry[i]=decAry[i]+32;

cout<<decAry[i];

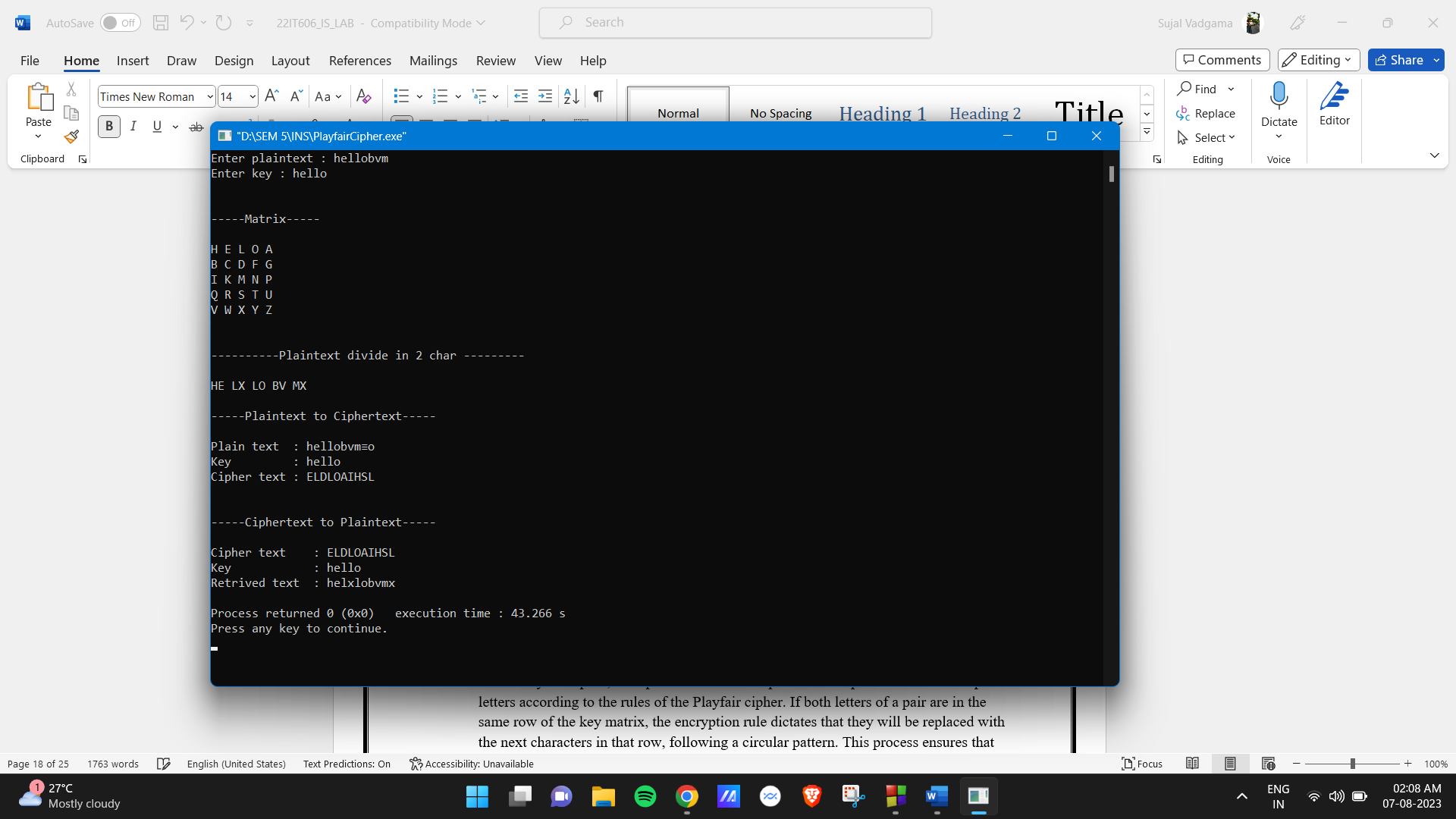
}

cout<<endl;

return 0;

}

**Output:**



**Conclusion:**

* In the Playfair cipher, each pair of letters in the plaintext is replaced with another pair of letters according to the rules of the Playfair cipher. If both letters of a pair are in the same row of the key matrix, the encryption rule dictates that they will be replaced with the next characters in that row, following a circular pattern. This process ensures that each digraph (pair of letters) in the plaintext is encrypted uniquely, adding an extra layer of security to the ciphertext.

**For ex.** EX is replaced with LW

* And in decryption each letter again replaces with another letter and create plaintext using matrix as per shown ciphertext pair is replace with another character from matrix as per the rules of decryption of Playfair cipher.

**For ex**. LW is replace with EX

**Practical – 4**

**Aim: To implement Polyalphabetic cipher encryption-decryption.**

**Code:**

#include<iostream>

using namespace std;

void vegenere(){

int kSize,pSize,cipher,p,k,pTemp,kTemp,choice=0;

char plainText[30],key[20],cipherText[30];

cout<<"------Vigenere cipher-----\n\n";

cout<<"Enter PlainText : ";

cin>>plainText;

cout<<"Enter key : ";

cin>>key;

for(pSize=0; plainText[pSize]!=NULL; pSize++);

for(kSize=0; key[kSize]!=NULL; kSize++);

pTemp=pSize;

kTemp=kSize;

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

{

if(plainText[i]>='A' && plainText[i]<='Z')

{

plainText[i]=plainText[i]+32;

}

}

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

{

if(key[i]>='A' && key[i]<='Z')

{

key[i]=key[i]+32;

}

}

if(pSize<kSize)

{

for(int i=pSize,ptr=0; i!=kSize; i++)

{

plainText[i]=plainText[ptr++];

}

pSize=kSize;

}

else if(kSize<pSize)

{

for(int i=kSize,ptr=0; i!=pSize; i++)

{

key[i]=key[ptr++];

}

kSize=pSize;

}

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

{

p=plainText[i]-97;

k=key[i]-97;

cipher=(p+k)%26;

cipherText[i]=cipher+65;

}

cout<<"\n\n-----Plaintext to Ciphertext-----"<<endl<<endl;

cout<<"Plain text : ";

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

{

cout<<plainText[i];

}

cout<<endl;

cout<<"key : ";

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

{

cout<<key[i];

}

cout<<endl;

cout<<"Cipher text : ";

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

{

cout<<cipherText[i];

}

cout<<endl<<endl;

cout<<"\n\n-----CipherText to Plaintext-----"<<endl<<endl;

cout<<"Cipher text : ";

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

{

cout<<cipherText[i];

}

cout<<endl;

cout<<"key : ";

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

{

cout<<key[i];

}

cout<<endl;

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

{

p=cipherText[i]-65;

k=key[i]-97;

cipher=(p-k)%26;

cipherText[i]=cipher+97;

}

cout<<"Plain text : ";

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

{

cout<<plainText[i];

}

cout<<endl<<endl;

}

void vernum(){

int kSize,pSize,cipher,p,k,pTemp,kTemp,plainText[30],key[20],cipherText[30];

cout<<"------Vernam cipher-----\n\n";

cout<<"Enter size of plaintext : ";

cin>>pSize;

pTemp=pSize;

for(int i=0;i<pSize;i++){

cout<<"Enter bit "<<(i+1)<<" : ";

cin>>plainText[i];

}

cout<<"Enter size of key : ";

cin>>kSize;

kTemp=kSize;

for(int i=0;i<kSize;i++){

cout<<"Enter bit "<<(i+1)<<" : ";

cin>>key[i];

}

if(pSize<kSize)

{

for(int i=pSize,ptr=0; i!=kSize; i++)

{

plainText[i]=plainText[ptr++];

}

pSize=kSize;

}

else if(kSize<pSize)

{

for(int i=kSize,ptr=0; i!=pSize; i++)

{

key[i]=key[ptr++];

}

kSize=pSize;

}

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

{

cipherText[i]=plainText[i]^key[i];

}

cout<<"\n\n-----Plaintext to Ciphertext-----"<<endl<<endl;

cout<<"Plain text : ";

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

{

cout<<plainText[i];

}

cout<<endl;

cout<<"key : ";

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

{

cout<<key[i];

}

cout<<endl;

cout<<"Cipher text : ";

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

{

cout<<cipherText[i];

}

cout<<endl<<endl;

cout<<"\n\n-----CipherText to Plaintext-----"<<endl<<endl;

cout<<"Cipher text : ";

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

{

cout<<cipherText[i];

}

cout<<endl;

cout<<"key : ";

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

{

cout<<key[i];

}

cout<<endl;

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

{

cipherText[i]=cipherText[i]^key[i];

}

cout<<"Plain text : ";

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

{

cout<<plainText[i];

}

cout<<endl<<endl;

}

int main(){

cout<<"--------Poly-alphabetic Cipher------\n\n";

cout<<"1] for vegenere ciphet\n2] for vernum cipher\nyou choice : ";

int choice;

cin>>choice;

if(choice==1){

vegenere();

}else if(choice==2){

vernum();

}else{

cout<<"Inavlid choice";

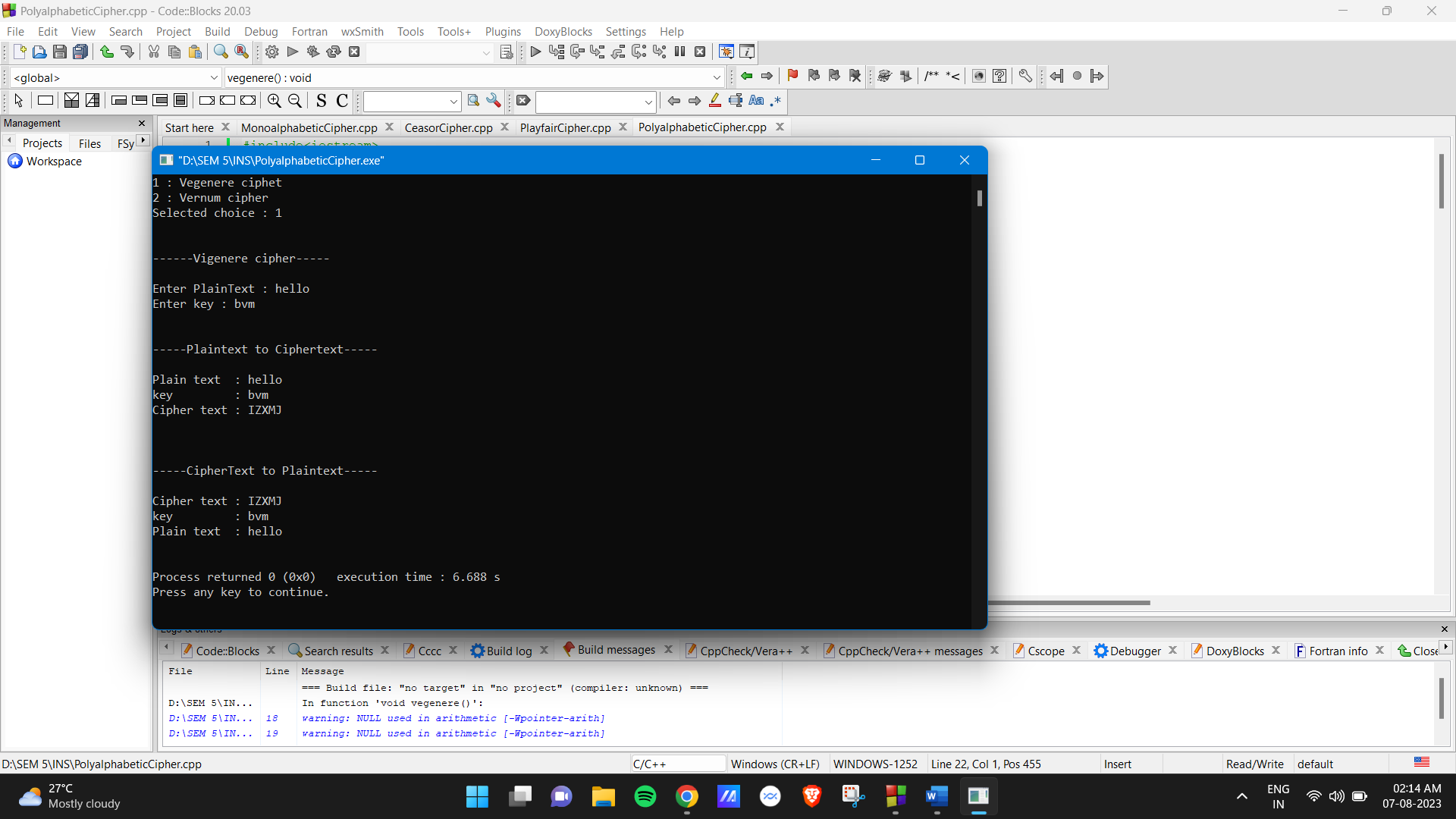
}

return 0;

}

**Output:**

**Vigenere Cipher:**



**Conclusion:**

* In Vegenere cipher each letter is exactly replace with another letter as shown in figure first if key is small than plaintext it is repeated and both will become of same length and then using formula plaintext is converted in ciphertext c = (p + k) mod 26.

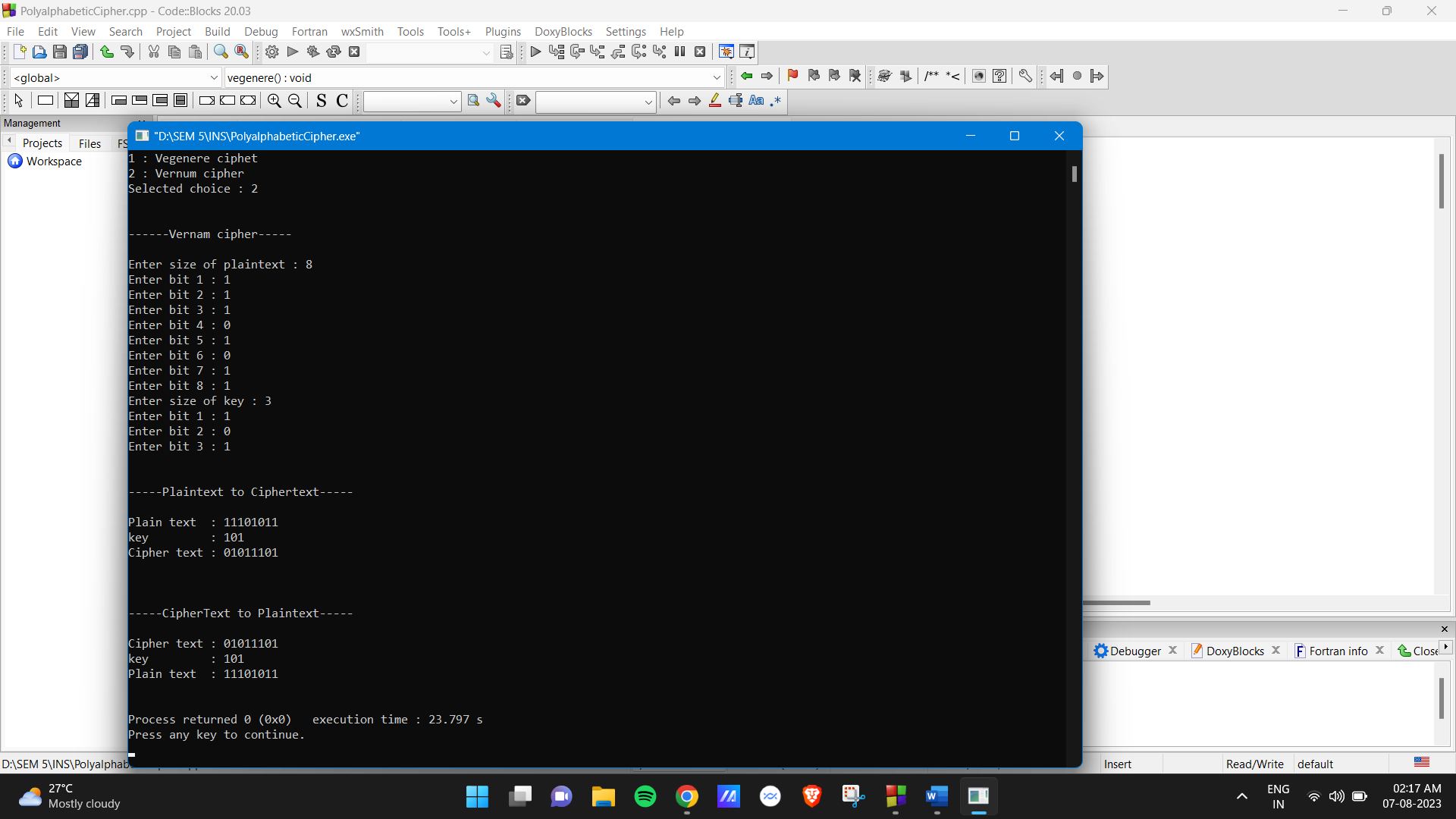
**For ex.** Plaintext: W, key : D, ciphertext = w(22) + D(3) mod 26 = 25(Z)

So, W is replaced with Z

* And in decryption each letter again replaces with another letter as per shown same as encryption process just use formula p = (c - k) mod 26.

**For ex**. Z is replace with W

**Vernam Cipher:**



**Conclusion:**

* In Vernam cipher each letter is exactly replace with another letter as shown in figure first if key is small than plaintext it is repeated and both will become of same length and then using xor operation on plaintext & key for ciphertext.

**For ex.** Plaintext : 1, Key : 1 = Ciphertext : 0

* And in decryption each letter again replaces with another letter as per shown same as encryption process xor operation is performed on ciphertext & key for generate plaintext.

**For ex**. Ciphertext : 0, Key : 1 = Plaintext : 1

**Practical – 5**

**Aim : To implement Columnar transposition cipher encryption-decryption.**

**Code:**

#include<iostream>

using namespace std;

int main()

{

char plaintext[100], key[100], ciphertext[100];

int plaintext\_count = 0, key\_count = 0, k = 1, row\_count = 0, p = 0, index = 0;

cout<<"-----Coloumn Transposition Cipher-----";

cout<<"\n\nEnter PlainText (Without Space) : ";

cin>>plaintext;

cout<<"Enter Key : ";

cin>>key;

for(int i = 0; plaintext[i] != NULL; i++)

{

if(plaintext[i] >= 'A' && plaintext[i] <= 'Z')

{

plaintext[i] = plaintext[i] + 32;

}

plaintext\_count++;

}

for(int i = 0; key[i] != NULL; i++)

{

if(key[i] >= 'A' && key[i] <= 'Z')

{

key[i] = key[i] + 32;

}

key\_count++;

}

char cipher[key\_count][100];

cout<<"\n\n.....ENCRYPTION.....\n";

cout<<"Plain Text : ";

for(int i = 0; i < plaintext\_count; i++)

{

cout<<plaintext[i];

}

cout<<"\nKey : ";

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

{

cout<<key[i];

}

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

{

cipher[0][i] = key[i];

}

int r;

for(r = 0; p < plaintext\_count; r++)

{

cipher[k][r] = plaintext[p] - 32;

p++;

if(r == key\_count - 1)

{

k++;

r = -1;

}

}

for(int i = r; r != 0 && i < key\_count; i++)

{

cipher[k][i] = 'X';

}

cout<<"\n\nTable for encryption : \n-----------------------\n";

if(r == 0) k--;

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

{

for(int j = 0; j < key\_count; j++)

{

cout<<cipher[i][j]<<" ";

}

cout<<"\n";

}

for(int i = 48; i <= 57 ; i++)

{

for(int j = 0; j < key\_count; j++)

{

if(cipher[0][j] == i)

{

for(int s = 1; s <= k; s++)

{

ciphertext[index] = cipher[s][j];

index++;

}

}

}

}

cout<<"\n\nCipher Text : ";

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

{

cout<<ciphertext[i];

}

cout<<"\n\n";

return 0;

}

**Output:**

A screenshot of a computer

Description automatically generated

**Conclusion:**

* In Columnar technique each letter is change letters position and generate ciphertext it cannot decrypt. In this technique each letter is change position with arrange in matrix and then generate ciphertext by column wise reading characters as per key.

**For ex.** S’s position in plaintext is 9th, replace position in ciphertext is at 15th position.

**Practical – 6**

**Aim : To implement Rail-Fence cipher encryption decryption.**

**Code:**

#include <iostream>

using namespace std;

string encryption(string plain\_text , int depth)

{

int length=plain\_text.length();

string cipher\_text="";

int index=0;

int main\_jump=(depth-1)\*2;

int minus\_two\_jump = main\_jump + 2;

int alter\_jump;

int k=0;

for(int level=1 ; level<=depth ; level++)

{

k=0;

//cout<<"level="<<level<<endl;

index=level-1;

minus\_two\_jump-=2;

alter\_jump = main\_jump - minus\_two\_jump;

while( index <= length-1 )

{

cipher\_text+=plain\_text[index];

//cout<<"i="<<index<<" ch="<<plain\_text[index];

if(level==1 || level==depth)

{

index+=main\_jump;

//cout<<" next i="<<index<< "main\_jump="<<main\_jump<<endl;

}

else

{

if(k==0)

{

index+=minus\_two\_jump;

//cout<<" next i="<<index<<" minus\_two\_jump="<<minus\_two\_jump<<endl;

k=!k;

}

else if(k==1)

{

index+=alter\_jump;

//cout<<" next i="<<index<<" alter\_jump="<<alter\_jump<<endl;;

k=!k;

}

}

}

}

return cipher\_text;

}

int main()

{

string plain\_text;

cout<<"------Rail Fence------\n";

cout<<"Enter Plain-Text(String with No space): ";

cin>>plain\_text;

int depth;

cout<<"Enter Depth:(If Depth=1 => Ceipher-Text= Plain-Text): ";

cin>>depth;

string ceipher\_text=encryption(plain\_text , depth);

cout<<"------Encryption------\n";

cout<<"Ceipher-Text: "<<ceipher\_text<<endl;

return 0;

}

**Output:**

A screenshot of a computer

Description automatically generated

**Conclusion:**

* In Rail Fence technique each letter is just replace the positions of the plaintext character and at last ciphertext is generated. In above figure the plaintext characters letter is not replace it is just change it’s position.

**For ex.** R is at 5th position and then after perform technique the P is in ciphertext at 2nd position.