// A C++ program to illustrate Caesar Cipher Technique

#include <iostream>

using namespace std;

// This function receives text and shift and

// returns the encrypted text

string encrypt(string text, int s)

{

string result = "";

// traverse text

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

{

// apply transformation to each character

// Encrypt Uppercase letters

if (isupper(text[i]))

result += char(int(text[i]+s-65)%26 +65);

// Encrypt Lowercase letters

else

result += char(int(text[i]+s-97)%26 +97);

}

// Return the resulting string

return result;

}

// Driver program to test the above function

int main()

{

string text;

int s;

cout << "Enter Text : ";

cin>>text;

cout << "\nEnter Key: ";

cin>>s;

cout << "\nCipher: " << encrypt(text, s);

return 0;

}

#include<iostream>

#include<string>

using namespace std;

int main(){

int i,j,k;

string s,t;

int key;

cout<<"Enter the key\n";

cin>>key;

cout<<"Enter the message to decrypt\n";

cin>>s;

for(i=0;i<s.size();i++){

t+=(s[i]-'A'-key+26)%26+'A';

}

cout<<"\n\nDecrypted message is "<<t<<'\n';

return 0;

}

#include <iostream>

#include<vector>

using namespace std;

void encrypt(int cipherMatrix[][1], vector<vector<int> > keyMatrix, int messageVector[][1], int m)

{

int x, i, j;

for (i = 0; i < m; i++)

{

for (j = 0; j < 1; j++)

{

cipherMatrix[i][j] = 0;

for (x = 0; x < m; x++)

{

cipherMatrix[i][j] += keyMatrix[i][x] \* messageVector[x][j];

}

cipherMatrix[i][j] = cipherMatrix[i][j] % 26;

}

}

string CipherText; //Converting the cipher vector into string

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

{

CipherText += cipherMatrix[i][0] + 65;

}

cout<<endl;

cout << "Ciphertext:" << CipherText;

}

int main()

{

int n;

cout<<"Enter the number of characters in string"<<endl;

cin>>n;

string message;

string key;

cout<<"Enter the string"<<endl;

cin>>message;

if(message.length()>n || message.length()<n)

{

cout<<endl;

cout<<"Length of string is not equal to the desired length"<<endl;

return 0;

}

cout<<"Enter the key of characters"<<endl;

cin>>key;

if(key.length()>(n\*n) || key.length()<(n\*n))

{

cout<<endl;

cout<<"Length of key is not equal to the desired length"<<endl;

return 0;

}

vector<vector<int> > keyMatrix(n, vector<int> (n)); //Converting the key string into matrix

int k = 0;

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

{

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

{

keyMatrix[i][j] = (key[k]) % 65;

k++;

}

}

int messageVector[n][1]; //Converting the message string into vector so that we are able to multiply

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

{

messageVector[i][0] = (message[i]) % 65;

}

int cipherMatrix[n][1]; //Encrypting the message

encrypt(cipherMatrix, keyMatrix, messageVector,n);

return 0;

}

#include<iostream>

#include<vector>

using namespace std;

int modInverse(int a, int m){

a=a%m;

for(int x=-m;x<m;x++)

if((a\*x)%m==1)

return x;

}

void getCofactor(vector<vector<int> > &a, vector<vector<int> > &temp, int p, int q, int n){

int i=0,j=0;

for(int row=0;row<n;row++){

for(int col=0;col<n;col++){

if(row!=p&&col!=q){

temp[i][j++] = a[row][col];

if (j==n-1){

j=0;

i++;

}

}

}

}

}

int determinant(vector<vector<int> > &a, int n, int N){

int D = 0;

if(n==1)

return a[0][0];

vector<vector<int> > temp(N, vector<int>(N));

int sign = 1;

for(int f=0;f<n;f++){

getCofactor(a, temp, 0, f, n);

D += sign \* a[0][f] \* determinant(temp, n - 1, N);

sign = -sign;

}

return D;

}

void adjoint(vector<vector<int> > &a,vector<vector<int> > &adj,int N){

if(N == 1){

adj[0][0] = 1;

return;

}

int sign = 1;

vector<vector<int> > temp(N, vector<int>(N));

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

for(int j=0;j<N;j++){

getCofactor(a, temp, i, j, N);

sign = ((i+j)%2==0)? 1: -1;

adj[j][i] = (sign)\*(determinant(temp, N-1 , N));

}

}

}

bool inverse(vector<vector<int> > &a, vector<vector<int> > &inv, int N){

int det = determinant(a, N, N);

if(det == 0){

cout << "Inverse does not exist";

return false;

}

int invDet = modInverse(det,26);

cout<<det%26<<' '<<invDet<<'\n';

vector<vector<int> > adj(N, vector<int>(N));

adjoint(a, adj, N);

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

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

inv[i][j] = (adj[i][j]\*invDet)%26;

return true;

}

int main(){

int x,y,i,j,k,n;

cout<<"Enter the size of key matrix\n";

cin>>n;

cout<<"Enter the key matrix\n";

vector<vector<int> > a(n, vector<int>(n));

vector<vector<int> > adj(n, vector<int>(n));

vector<vector<int> > inv(n, vector<int>(n));

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

for(j=0;j<n;j++){

cin>>a[i][j];

}

}

if(inverse(a,inv,n)){

cout<<"Inverse exist\n";

}

cout<<"Enter the message to decrypt\n";

string s;

cin>>s;

k=0;

string ans;

while(k<s.size()){

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

int sum = 0;

int temp = k;

for(j=0;j<n;j++){

sum += ((inv[i][j] + 26)%26\*(s[temp++]-'a')%26)%26;

sum = sum%26;

}

ans+=(sum+'a');

}

k+=n;

}

//ans+='\0';

int f=ans.size()-1;

while(ans[f]=='x'){

f--;

}

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

cout<<ans[i];

}

cout<<'\n';

return 0;

}

// C program to implement Playfair Cipher

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define SIZE 30

// Function to convert the string to lowercase

void toLowerCase(char plain[], int ps)

{

int i;

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

if (plain[i] > 64 && plain[i] < 91)

plain[i] += 32;

}

}

// Function to remove all spaces in a string

int removeSpaces(char\* plain, int ps)

{

int i, count = 0;

for (i = 0; i < ps; i++)

if (plain[i] != ' ')

plain[count++] = plain[i];

plain[count] = '\0';

return count;

}

// Function to generate the 5x5 key square

void generateKeyTable(char key[], int ks, char keyT[5][5])

{

int i, j, k, flag = 0, \*dicty;

// a 26 character hashmap

// to store count of the alphabet

dicty = (int\*)calloc(26, sizeof(int));

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

if (key[i] != 'j')

dicty[key[i] - 97] = 2;

}

dicty['j' - 97] = 1;

i = 0;

j = 0;

for (k = 0; k < ks; k++) {

if (dicty[key[k] - 97] == 2) {

dicty[key[k] - 97] -= 1;

keyT[i][j] = key[k];

j++;

if (j == 5) {

i++;

j = 0;

}

}

}

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

if (dicty[k] == 0) {

keyT[i][j] = (char)(k + 97);

j++;

if (j == 5) {

i++;

j = 0;

}

}

}

}

// Function to search for the characters of a digraph

// in the key square and return their position

void search(char keyT[5][5], char a, char b, int arr[])

{

int i, j;

if (a == 'j')

a = 'i';

else if (b == 'j')

b = 'i';

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

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

if (keyT[i][j] == a) {

arr[0] = i;

arr[1] = j;

}

else if (keyT[i][j] == b) {

arr[2] = i;

arr[3] = j;

}

}

}

}

// Function to find the modulus with 5

int mod5(int a) { return (a % 5); }

// Function to make the plain text length to be even

int prepare(char str[], int ptrs)

{

if (ptrs % 2 != 0) {

str[ptrs++] = 'z';

str[ptrs] = '\0';

}

return ptrs;

}

// Function for performing the encryption

void encrypt(char str[], char keyT[5][5], int ps)

{

int i, a[4];

for (i = 0; i < ps; i += 2) {

search(keyT, str[i], str[i + 1], a);

if (a[0] == a[2]) {

str[i] = keyT[a[0]][mod5(a[1] + 1)];

str[i + 1] = keyT[a[0]][mod5(a[3] + 1)];

}

else if (a[1] == a[3]) {

str[i] = keyT[mod5(a[0] + 1)][a[1]];

str[i + 1] = keyT[mod5(a[2] + 1)][a[1]];

}

else {

str[i] = keyT[a[0]][a[3]];

str[i + 1] = keyT[a[2]][a[1]];

}

}

}

// Function to encrypt using Playfair Cipher

void encryptByPlayfairCipher(char str[], char key[])

{

char ps, ks, keyT[5][5];

// Key

ks = strlen(key);

ks = removeSpaces(key, ks);

toLowerCase(key, ks);

// Plaintext

ps = strlen(str);

toLowerCase(str, ps);

ps = removeSpaces(str, ps);

ps = prepare(str, ps);

generateKeyTable(key, ks, keyT);

encrypt(str, keyT, ps);

}

// Driver code

int main()

{

char str[SIZE], key[SIZE];

// Key to be encrypted

strcpy(key, "Monarchy");

printf("Key text: %s\n", key);

// Plaintext to be encrypted

strcpy(str, "instruments");

printf("Plain text: %s\n", str);

// encrypt using Playfair Cipher

encryptByPlayfairCipher(str, key);

printf("Cipher text: %s\n", str);

return 0;

}

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define SIZE 30

// Convert all the characters

// of a string to lowercase

void toLowerCase(char plain[], int ps)

{

int i;

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

if (plain[i] > 64 && plain[i] < 91)

plain[i] += 32;

}

}

// Remove all spaces in a string

// can be extended to remove punctuation

int removeSpaces(char\* plain, int ps)

{

int i, count = 0;

for (i = 0; i < ps; i++)

if (plain[i] != ' ')

plain[count++] = plain[i];

plain[count] = '\0';

return count;

}

// generates the 5x5 key square

void generateKeyTable(char key[], int ks, char keyT[5][5])

{

int i, j, k, flag = 0, \*dicty;

// a 26 character hashmap

// to store count of the alphabet

dicty = (int\*)calloc(26, sizeof(int));

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

if (key[i] != 'j')

dicty[key[i] - 97] = 2;

}

dicty['j' - 97] = 1;

i = 0;

j = 0;

for (k = 0; k < ks; k++) {

if (dicty[key[k] - 97] == 2) {

dicty[key[k] - 97] -= 1;

keyT[i][j] = key[k];

j++;

if (j == 5) {

i++;

j = 0;

}

}

}

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

if (dicty[k] == 0) {

keyT[i][j] = (char)(k + 97);

j++;

if (j == 5) {

i++;

j = 0;

}

}

}

}

// Search for the characters of a digraph

// in the key square and return their position

void search(char keyT[5][5], char a, char b, int arr[])

{

int i, j;

if (a == 'j')

a = 'i';

else if (b == 'j')

b = 'i';

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

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

if (keyT[i][j] == a) {

arr[0] = i;

arr[1] = j;

}

else if (keyT[i][j] == b) {

arr[2] = i;

arr[3] = j;

}

}

}

}

// Function to find the modulus with 5

int mod5(int a)

{

if (a < 0)

a += 5;

return (a % 5);

}

// Function to decrypt

void decrypt(char str[], char keyT[5][5], int ps)

{

int i, a[4];

for (i = 0; i < ps; i += 2) {

search(keyT, str[i], str[i + 1], a);

if (a[0] == a[2]) {

str[i] = keyT[a[0]][mod5(a[1] - 1)];

str[i + 1] = keyT[a[0]][mod5(a[3] - 1)];

}

else if (a[1] == a[3]) {

str[i] = keyT[mod5(a[0] - 1)][a[1]];

str[i + 1] = keyT[mod5(a[2] - 1)][a[1]];

}

else {

str[i] = keyT[a[0]][a[3]];

str[i + 1] = keyT[a[2]][a[1]];

}

}

}

// Function to call decrypt

void decryptByPlayfairCipher(char str[], char key[])

{

char ps, ks, keyT[5][5];

// Key

ks = strlen(key);

ks = removeSpaces(key, ks);

toLowerCase(key, ks);

// ciphertext

ps = strlen(str);

toLowerCase(str, ps);

ps = removeSpaces(str, ps);

generateKeyTable(key, ks, keyT);

decrypt(str, keyT, ps);

}

// Driver code

int main()

{

char str[SIZE], key[SIZE];

// Key to be encrypted

strcpy(key, "Monarchy");

printf("Key text: %s\n", key);

// Ciphertext to be decrypted

strcpy(str, "gatlmzclrqtx");

printf("Plain text: %s\n", str);

// encrypt using Playfair Cipher

decryptByPlayfairCipher(str, key);

printf("Deciphered text: %s\n", str);

return 0;

}