**1] Caesar Cipher**  
  
#include<bits/stdc++.h>

using namespace std;

int main()

{

cout << "Enter the mode, Encrypt - 0, Decrypt - 1" << endl;

int mode;

cin >> mode;

if(mode == 0)

{

cout << "Enter the Plain Text: ";

string plain;

cin >> plain;

for(int key = 1; key <= 25; ++key)

{

for(auto &c : plain)

{

int index = c - 'a';

index = (index + key) % 26;

cout << (char)('a' + index);

}

cout << endl;

}

}

else if(mode == 1){

cout << "Enter the Cipher Text: ";

string cipher;

cin >> cipher;

for(int key = 1; key <= 25; ++key)

{

for(auto &c : cipher)

{

int index = c - 'a';

index = (index - key);

if(index < 0)index += 26;

cout << (char)('a' + index);

}

cout << endl;

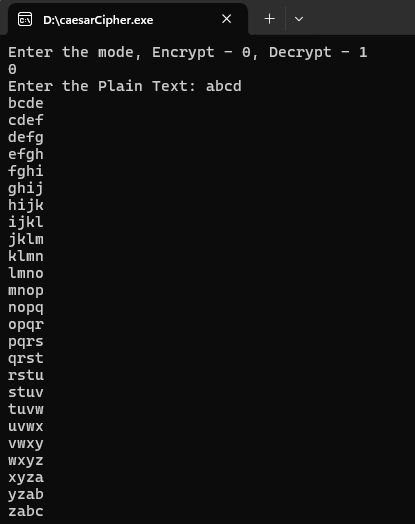
}

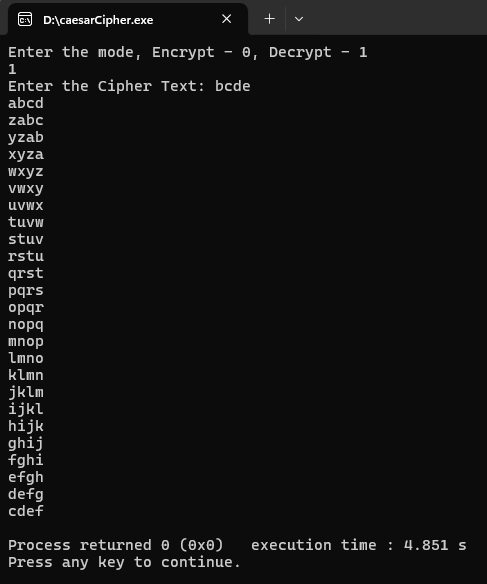
}

return 0;

}

**OUTPUT:**





**2] Play Fair:**

A] Without key

*#include <bits/stdc++.h>*

*using namespace std;*

*void initMatrix(vector<vector<char>>& matrix)*

*{*

*char c = 'a';*

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

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

*if (c == 'j')*

*c++;*

*matrix[i][j] = c;*

*c++;*

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

*}*

*cout << endl;*

*}*

*}*

*string preprocessPlainText(string plain, const string& padding) {*

*// Replace 'j' with 'i'*

*for (int i = 0; i < plain.size(); i++) {*

*if (plain[i] == 'j') {*

*plain[i] = 'i';*

*}*

*}*

*for (int i = 0; i < plain.size() - 1; i++) {*

*if (plain[i] == plain[i + 1]) {*

*plain.insert(i + 1, padding);*

*i++;*

*}*

*}*

*if (plain.size() % 2 == 1) {*

*plain += padding;*

*}*

*return plain;*

*}*

*vector<pair<char, char>> divideText(const string& plain)*

*{*

*vector<pair<char, char>> text;*

*for (int i = 0; i < plain.size(); i += 2) {*

*text.push\_back({plain[i], plain[i + 1]});*

*}*

*return text;*

*}*

*string processText(const vector<pair<char, char>>& text, const vector<vector<char>>& matrix, bool encrypt) {*

*string resultText = "";*

*int r1, r2, c1, c2;*

*for (const auto& pair : text) {*

*// Find positions of the characters in the matrix*

*for (int a = 0; a < 5; ++a) {*

*for (int b = 0; b < 5; ++b) {*

*if (matrix[a][b] == pair.first) {*

*r1 = a;*

*c1 = b;*

*}*

*if (matrix[a][b] == pair.second) {*

*r2 = a;*

*c2 = b;*

*}*

*}*

*}*

*if (r1 == r2) {*

*resultText += matrix[r1][(c1 + (encrypt ? 1 : 4)) % 5];*

*resultText += matrix[r2][(c2 + (encrypt ? 1 : 4)) % 5];*

*} else if (c1 == c2) {*

*resultText += matrix[(r1 + (encrypt ? 1 : 4)) % 5][c1];*

*resultText += matrix[(r2 + (encrypt ? 1 : 4)) % 5][c2];*

*} else {*

*resultText += matrix[r1][c2];*

*resultText += matrix[r2][c1];*

*}*

*}*

*return resultText;*

*}*

*string removePadding(string& plainText, const string& padding) {*

*cout << plainText << endl;*

*if (!plainText.empty() && plainText.back() == padding.back()) {*

*plainText.erase(plainText.size() - 1);*

*}*

*return plainText;*

*}*

*int main() {*

*vector<vector<char>> matrix(5, vector<char>(5));*

*string padding = "x";*

*initMatrix(matrix);*

*cout << endl << "Enter the plainText: ";*

*string plain;*

*cin >> plain;*

*plain = preprocessPlainText(plain, padding);*

*cout << endl << "Plain Text: " << plain << endl;*

*vector<pair<char, char>> text = divideText(plain);*

*string cipherText = processText(text, matrix, true);*

*cout << endl << "Cipher Text: " << cipherText << endl;*

*cout << "Decryption" << endl;*

*vector<pair<char, char>> cipher = divideText(cipherText);*

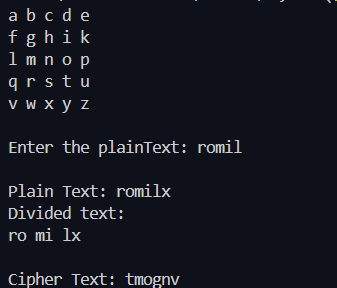
*string decryptedText = processText(cipher, matrix, false);*

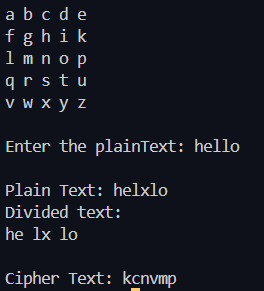
*decryptedText = removePadding(decryptedText, padding);*

*cout << decryptedText << endl;*

*return 0;*

*}*

**OUTPUT:**  




B] with Key

#include <bits/stdc++.h>

using namespace std;

void initMatrix(vector<vector<char>>& matrix, string& key) {

set<char> usedChars;

string uniqueChars;

for (char c : key)

{

if (c == 'j') c = 'i';

if (usedChars.find(c) == usedChars.end() && c >= 'a' && c <= 'z') {

usedChars.insert(c);

uniqueChars += c;

}

}

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

{

if (c == 'j') continue;

if (usedChars.find(c) == usedChars.end()) {

usedChars.insert(c);

uniqueChars += c;

}

}

int index = 0;

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

{

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

{

matrix[i][j] = uniqueChars[index++];

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

}

cout << endl;

}

}

string preprocessPlainText(string plain, const string& padding) {

// Replace 'j' with 'i'

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

if (plain[i] == 'j') {

plain[i] = 'i';

}

}

for (int i = 0; i < plain.size() - 1; i++) {

if (plain[i] == plain[i + 1]) {

plain.insert(i + 1, padding);

i++;

}

}

if (plain.size() % 2 == 1) {

plain += padding;

}

return plain;

}

vector<pair<char, char>> divideText(const string& plain)

{

vector<pair<char, char>> text;

for (int i = 0; i < plain.size(); i += 2) {

text.push\_back({plain[i], plain[i + 1]});

}

return text;

}

string processText(const vector<pair<char, char>>& text, const vector<vector<char>>& matrix, bool encrypt) {

string resultText = "";

int r1, r2, c1, c2;

for (const auto& pair : text) {

// Find positions of the characters in the matrix

for (int a = 0; a < 5; ++a) {

for (int b = 0; b < 5; ++b) {

if (matrix[a][b] == pair.first) {

r1 = a;

c1 = b;

}

if (matrix[a][b] == pair.second) {

r2 = a;

c2 = b;

}

}

}

if (r1 == r2) {

resultText += matrix[r1][(c1 + (encrypt ? 1 : 4)) % 5];

resultText += matrix[r2][(c2 + (encrypt ? 1 : 4)) % 5];

} else if (c1 == c2) {

resultText += matrix[(r1 + (encrypt ? 1 : 4)) % 5][c1];

resultText += matrix[(r2 + (encrypt ? 1 : 4)) % 5][c2];

} else {

resultText += matrix[r1][c2];

resultText += matrix[r2][c1];

}

}

return resultText;

}

string removePadding(string& plainText, const string& padding) {

cout << plainText << endl;

if (!plainText.empty() && plainText.back() == padding.back()) {

plainText.erase(plainText.size() - 1);

}

return plainText;

}

int main() {

vector<vector<char>> matrix(5, vector<char>(5));

string padding = "x";

string key = "monarchy";

initMatrix(matrix, key);

cout << endl << "Enter the plainText: ";

string plain;

cin >> plain;

plain = preprocessPlainText(plain, padding);

cout << endl << "Plain Text: " << plain << endl;

vector<pair<char, char>> text = divideText(plain);

string cipherText = processText(text, matrix, true);

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

cout << "Decryption" << endl;

vector<pair<char, char>> cipher = divideText(cipherText);

string decryptedText = processText(cipher, matrix, false);

decryptedText = removePadding(decryptedText, padding);

cout << decryptedText << endl;

return 0;

}

**OUTPUT:**

