**1] Rail Fence:**

#include <bits/stdc++.h>

using namespace std;

// Function to create the Rail Fence Matrix for encryption

vector<vector<char>> initMatrix(const string& plain, int n) {

int m = plain.size();

vector<vector<char>> matrix(n, vector<char>(m, '-'));

int r = 0, c = 0;

bool down = true;

for (char ch : plain) {

matrix[r][c++] = ch;

if (down) {

if (r == n - 1) {

down = false;

r--;

} else {

r++;

}

} else {

if (r == 0) {

down = true;

r++;

} else {

r--;

}

}

}

return matrix;

}

// Function to extract the cipher text from the Rail Fence Matrix

string cipherText(const vector<vector<char>>& matrix) {

string cipher = "";

for (const auto& row : matrix) {

for (char ch : row) {

if (ch != '-') {

cipher += ch;

}

}

}

return cipher;

}

// Function to create Rail Matrix for decryption

void decryptionMatrix(vector<vector<char>>& matrix, int n, int m) {

int r = 0, c = 0;

bool down = true;

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

matrix[r][c++] = '\*';

if (down) {

if (r == n - 1) {

down = false;

r--;

} else {

r++;

}

} else {

if (r == 0) {

down = true;

r++;

} else {

r--;

}

}

}

}

// Function to fill the Rail Matrix with the cipher text for decryption

void fillDecryptionMatrix(vector<vector<char>>& matrix, const string& cipher) {

int k = 0;

for (auto& row : matrix) {

for (char& ch : row) {

if (ch == '\*') {

ch = cipher[k++];

}

}

}

}

// Function to decrypt the text from the Rail Matrix

string decryptionText(const vector<vector<char>>& matrix, int n, int m) {

string decryptedText = "";

int r = 0, c = 0;

bool down = true;

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

decryptedText += matrix[r][c++];

if (down) {

if (r == n - 1) {

down = false;

r--;

} else {

r++;

}

} else {

if (r == 0) {

down = true;

r++;

} else {

r--;

}

}

}

return decryptedText;

}

int main() {

string plain;

cout << "Enter the plain text: ";

cin >> plain;

int n;

cout << "Enter the key: ";

cin >> n;

// Encryption

vector<vector<char>> matrix = initMatrix(plain, n);

cout << "\nRail Fence Matrix (Encryption):\n";

for (const auto& row : matrix) {

for (char ch : row) {

cout << ch;

}

cout << endl;

}

string cipher = cipherText(matrix);

cout << "\nCipher Text: " << cipher << endl;

// Decryption

int m = plain.size();

decryptionMatrix(matrix, n, m);

cout << "\nRail Fence Matrix (Marked for Decryption):\n";

for (const auto& row : matrix) {

for (char ch : row) {

cout << ch;

}

cout << endl;

}

fillDecryptionMatrix(matrix, cipher);

cout << "\nRail Fence Matrix (Filled with Cipher Text):\n";

for (const auto& row : matrix) {

for (char ch : row) {

cout << ch;

}

cout << endl;

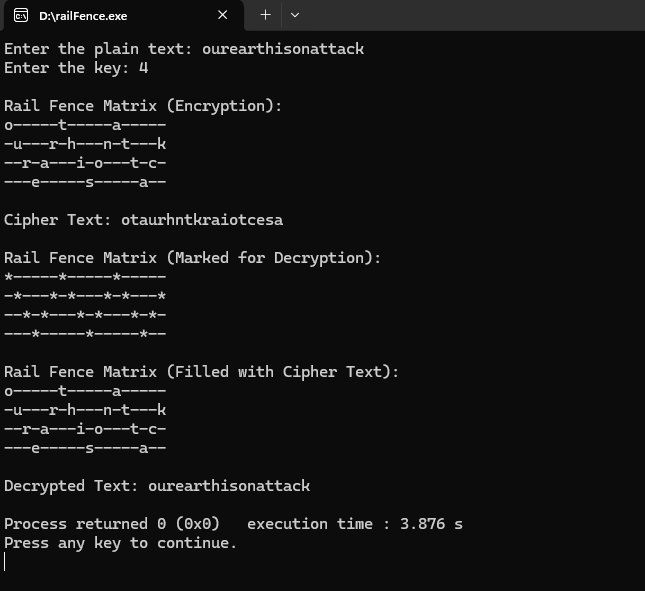
}

string decryptedText = decryptionText(matrix, n, m);

cout << "\nDecrypted Text: " << decryptedText << endl;

return 0;

}

**OUTPUT:**  


**2] Columnar with key:**

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

using namespace std;

*// Function to create the Columnar Matrix for encryption*

vector<vector<char>> initMatrix(const string &*plain*, string &*keyword*)

{

int index = 0;

int p = *plain*.size();

int k = *keyword*.size();

int n = (p + k - 1) / k;

vector<vector<char>> v(n, vector<char>(k, '\_'));

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

{

*for* (int j = 0; j < k; ++j)

{

*if* (index < p)

v[i][j] = *plain*[index++];

}

}

*return* v;

}

vector<pair<char, int>> indices(string &*keyword*, bool *decrypt*)

{

vector<pair<char, int>> v;

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

{

v.push\_back({*keyword*[i], i});

}

sort(v.begin(), v.end(), [](const pair<char, int> &*a*, const pair<char, int> &*b*)

{ *return* *a*.first < *b*.first; });

*return* v;

}

vector<vector<char>> rearrange(const vector<pair<char, int>> &*indx*, vector<vector<char>> &*matrix*, bool *decrypt*)

{

int row = *matrix*.size();

int col = *matrix*[0].size();

vector<vector<char>> rearranged(row, vector<char>(col, '\_'));

*for* (int c = 0; c < col; ++c)

{

int sortedIndex = *indx*[c].second;

*for* (int r = 0; r < row; ++r)

{

*if* (!*decrypt*)

{

rearranged[r][c] = *matrix*[r][sortedIndex];

}

*else*

{

rearranged[r][sortedIndex] = *matrix*[r][c];

}

}

}

*return* rearranged;

}

string encryptedText(vector<vector<char>> &*matrix*)

{

string cipherText = "";

*for* (auto &i : *matrix*)

{

*for* (auto &j : i)

{

cipherText += j;

}

}

*return* cipherText;

}

vector<vector<char>> cipherTextToDmatrix(string &*cipherText*, string &*keyword*)

{

int index = 0;

int k = *keyword*.size();

int c = *cipherText*.size();

int n = (c + k - 1) / k;

vector<vector<char>> v(n, vector<char>(k, '\_'));

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

{

*for* (int j = 0; j < k; ++j)

{

*if* (index < c)

v[i][j] = *cipherText*[index++];

}

}

*return* v;

}

string decryptionText(vector<vector<char>> &*decryptionMatrix*)

{

string decryptedText = "";

*for* (auto &i : *decryptionMatrix*)

{

*for* (auto j : i)

{

*if* (j != '\_')

decryptedText += j;

}

}

*return* decryptedText;

}

int main()

{

string plain;

cout << "Enter the plain text: ";

cin >> plain;

string keyword;

cout << "Enter the keyword: ";

cin >> keyword;

*// Encryption*

vector<vector<char>> matrix = initMatrix(plain, keyword);

cout << endl

<< "Encryption:" << endl

<< endl

<< "Plain text to encryption matrix: " << endl;

*for* (auto &i : matrix)

{

*for* (auto &j : i)

{

cout << j << " ";

}

cout << endl;

}

bool decrypt = false;

vector<pair<char, int>> indx = indices(keyword, decrypt);

matrix = rearrange(indx, matrix, decrypt);

cout << endl

<< "Rearranged encryption matrix: " << endl;

*for* (auto &i : matrix)

{

*for* (auto &j : i)

{

cout << j << " ";

}

cout << endl;

}

string cipherText = encryptedText(matrix);

cout << endl

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

cout << endl

<< "-----------------------------------------------------" << endl;

*// Decryption*

vector<vector<char>> decryptionMatrix = cipherTextToDmatrix(cipherText, keyword);

cout << endl

<< "Decryption:" << endl;

cout << endl

<< "Cipher text to decryption matrix: " << endl;

*for* (auto &i : decryptionMatrix)

{

*for* (auto &j : i)

{

cout << j << " ";

}

cout << endl;

}

decrypt = true;

decryptionMatrix = rearrange(indx, decryptionMatrix, decrypt);

cout << endl

<< "Rearranged decryption matrix: " << endl;

*for* (auto &i : decryptionMatrix)

{

*for* (auto &j : i)

{

cout << j << " ";

}

cout << endl;

}

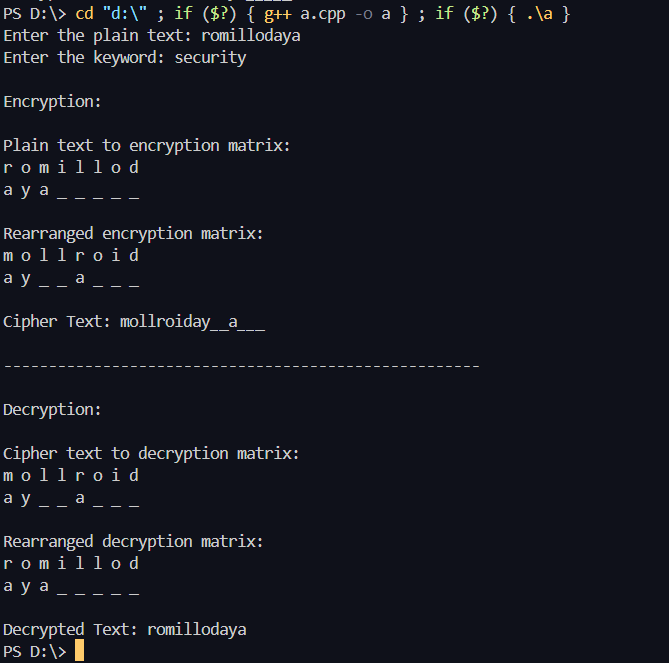
string decryptedText = decryptionText(decryptionMatrix);

cout << endl

<< "Decrypted Text: " << decryptedText << endl;

*return* 0;

}

**OUTPUT:**  


**3] Columnar without key:**

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

using namespace std;

*// Function to create the Columnar Matrix for encryption*

vector<vector<char>> initMatrix(const string &*plain*, int *n*)

{

int p = *plain*.size();

vector<vector<char>> v(*n*, vector<char>(*n*, '\_'));

*for* (int i = 0, j = 0, k = 0; k < p; ++j)

{

*if* (j == *n*)

{

j = 0;

i++;

}

v[i][j] = *plain*[k++];

}

*return* v;

}

string encryptedText(vector<vector<char>> &*matrix*, int *n*)

{

string cipherText = "";

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

{

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

{

cipherText += *matrix*[j][i];

}

}

*return* cipherText;

}

vector<vector<char>> cipherTextToDmatrix(string &*cipherText*, int *n*)

{

int c = *cipherText*.size();

vector<vector<char>> v(5, vector<char>(5, '\_'));

*for* (int i = 0, k = 0; i < *n*; ++i)

{

*for*(int j = 0; j < *n* && k < *cipherText*.size(); ++j)

{

v[j][i] = *cipherText*[k++];

}

}

*return* v;

}

string decryptionText(vector<vector<char>> &*decryptionMatrix*)

{

string decryptedText = "";

*for* (auto &i : *decryptionMatrix*)

{

*for* (auto j : i)

{

*if* (j != '\_')

decryptedText += j;

}

}

*return* decryptedText;

}

int main()

{

string plain;

cout << "Enter the plain text: ";

cin >> plain;

int n = 5;

*// Encryption*

vector<vector<char>> matrix = initMatrix(plain, n);

cout << endl

<< "Encryption:" << endl

<< endl

<< "Plain text to encryption matrix: " << endl;

*for* (auto &i : matrix)

{

*for* (auto &j : i)

{

cout << j << " ";

}

cout << endl;

}

string cipherText = encryptedText(matrix, n);

cout << endl

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

cout << endl

<< "-----------------------------------------------------" << endl;

*// Decryption*

vector<vector<char>> decryptionMatrix = cipherTextToDmatrix(cipherText, n);

cout << endl

<< "Decryption:" << endl;

cout << endl

<< "Cipher text to decryption matrix: " << endl;

*for* (auto &i : decryptionMatrix)

{

*for* (auto &j : i)

{

cout << j << " ";

}

cout << endl;

}

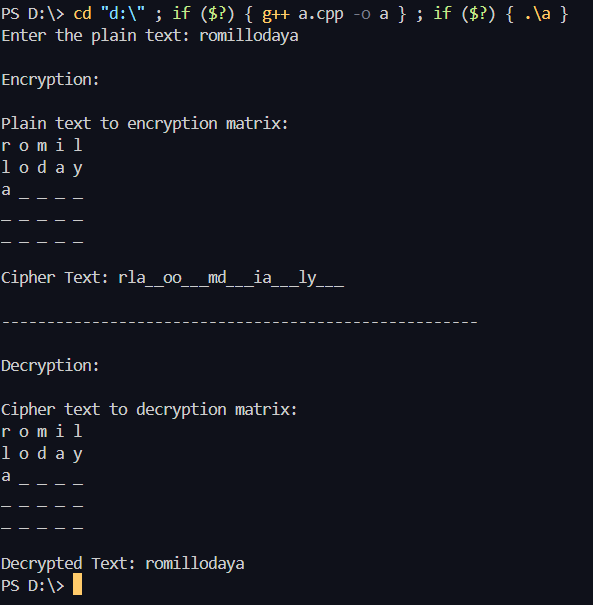
string decryptedText = decryptionText(decryptionMatrix);

cout << endl

<< "Decrypted Text: " << decryptedText << endl;

*return* 0;

}

**OUTPUT:  
  
**