**Cryptography & Network Security Lab**

**Assignment 02**

**Rail Fence Cipher:**

*// C++ program to illustrate Rail Fence Cipher*

*// Encryption and Decryption*

#include <bits/stdc++.h>

using namespace std;

*// function to encrypt a message*

string encryptRailFence(string *text*, int *key*)

{

*// create the matrix to cipher plain text*

*// key = rows , length(text) = columns*

    char rail[*key*][(*text*.length())];

*// filling the rail matrix to distinguish filled*

*// spaces from blank ones*

    for (int i=0; i < *key*; i++)

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

            rail[i][j] = '\n';

*// to find the direction*

    bool dir\_down = false;

    int row = 0, col = 0;

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

    {

*// check the direction of flow*

*// reverse the direction if we've just*

*// filled the top or bottom rail*

        if (row == 0 || row == *key*-1)

            dir\_down = !dir\_down;

*// fill the corresponding alphabet*

        rail[row][col++] = *text*[i];

*// find the next row using direction flag*

        dir\_down?row++ : row--;

    }

*//now we can construct the cipher using the rail matrix*

    string result;

    for (int i=0; i < *key*; i++)

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

            if (rail[i][j]!='\n')

                result.push\_back(rail[i][j]);

    return result;

}

*// This function receives cipher-text and key*

*// and returns the original text after decryption*

string decryptRailFence(string *cipher*, int *key*)

{

*// create the matrix to cipher plain text*

*// key = rows , length(text) = columns*

    char rail[*key*][*cipher*.length()];

*// filling the rail matrix to distinguish filled*

*// spaces from blank ones*

    for (int i=0; i < *key*; i++)

        for (int j=0; j < *cipher*.length(); j++)

            rail[i][j] = '\n';

*// to find the direction*

    bool dir\_down;

    int row = 0, col = 0;

*// mark the places with '\*'*

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

    {

*// check the direction of flow*

        if (row == 0)

            dir\_down = true;

        if (row == *key*-1)

            dir\_down = false;

*// place the marker*

        rail[row][col++] = '\*';

*// find the next row using direction flag*

        dir\_down?row++ : row--;

    }

*// now we can construct the fill the rail matrix*

    int index = 0;

    for (int i=0; i<*key*; i++)

        for (int j=0; j<*cipher*.length(); j++)

            if (rail[i][j] == '\*' && index<*cipher*.length())

                rail[i][j] = *cipher*[index++];

*// now read the matrix in zig-zag manner to construct*

*// the resultant text*

    string result;

    row = 0, col = 0;

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

    {

*// check the direction of flow*

        if (row == 0)

            dir\_down = true;

        if (row == *key*-1)

            dir\_down = false;

*// place the marker*

        if (rail[row][col] != '\*')

            result.push\_back(rail[row][col++]);

*// find the next row using direction flag*

        dir\_down?row++: row--;

    }

    return result;

}

*//driver program to check the above functions*

int main()

{

    string message;

    int rails;

    cout << "Enter a message: ";

    getline(cin, message);

    cout << "Enter the number of rails: ";

    cin >> rails;

    string encryptedMessage = encryptRailFence(message, rails);

    string decryptedMessage = decryptRailFence(encryptedMessage, rails);

    cout << "Encrypted message: " << encryptedMessage << endl;

    cout << "Decrypted message: " << decryptedMessage << endl;

    return 0;

}

**Result:**

**Columnar Cipher:**

#include <iostream>

#include <string>

#include <map>

using namespace std;

void setPermutationOrder(const string& *key*, map<char, int>& *keyMap*) {

    for (int i = 0; i < *key*.length(); i++) {

*keyMap*[*key*[i]] = i;

    }

}

string encryptMessage(const string& *msg*, const string& *key*, map<char, int>& *keyMap*) {

    int row, col, j;

    string cipher = "";

    col = *key*.length();

    row = *msg*.length() / col;

    if (*msg*.length() % col) {

        row += 1;

    }

    char matrix[row][col];

    for (int i = 0, k = 0; i < row; i++) {

        for (int j = 0; j < col; ) {

            if (*msg*[k] == '\0') {

                matrix[i][j] = '\_';

                j++;

            }

            if (isalpha(*msg*[k]) || *msg*[k] == ' ') {

                matrix[i][j] = *msg*[k];

                j++;

            }

            k++;

        }

    }

    for (map<char, int>::iterator ii = *keyMap*.begin(); ii != *keyMap*.end(); ++ii) {

        j = ii->second;

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

            if (isalpha(matrix[i][j]) || matrix[i][j] == ' ' || matrix[i][j] == '\_') {

                cipher += matrix[i][j];

            }

        }

    }

    return cipher;

}

string decryptMessage(const string& *cipher*, const string& *key*, map<char, int>& *keyMap*) {

    int col = *key*.length();

    int row = *cipher*.length() / col;

    char cipherMat[row][col];

    for (int j = 0, k = 0; j < col; j++) {

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

            cipherMat[i][j] = *cipher*[k++];

        }

    }

    int index = 0;

    for (map<char, int>::iterator ii = *keyMap*.begin(); ii != *keyMap*.end(); ++ii) {

        ii->second = index++;

    }

    char decCipher[row][col];

    for (int k = 0, l, j; *key*[k] != '\0'; k++) {

        l = *keyMap*[*key*[k]];

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

            decCipher[i][k] = cipherMat[i][l];

        }

    }

    string msg = "";

    for (int i = 0; i < row; i++) {

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

            if (decCipher[i][j] != '\_') {

                msg += decCipher[i][j];

            }

        }

    }

    return msg;

}

int main() {

    string message, key;

    cout << "Enter a message: ";

    getline(cin, message);

    cout << "Enter the encryption key: ";

    cin >> key;

    map<char, int> keyMap;

    setPermutationOrder(key, keyMap);

    string encryptedMessage = encryptMessage(message, key, keyMap);

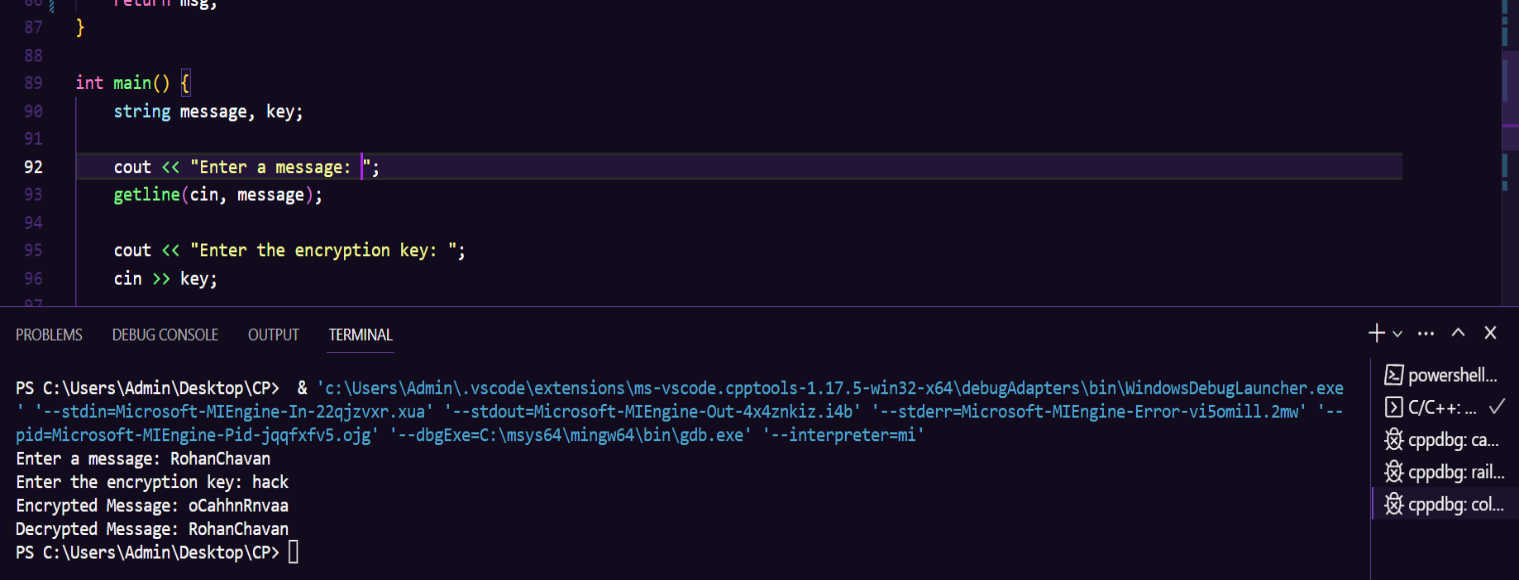
    string decryptedMessage = decryptMessage(encryptedMessage, key, keyMap);

    cout << "Encrypted Message: " << encryptedMessage << endl;

    cout << "Decrypted Message: " << decryptedMessage << endl;

    return 0;

}

**Results:**