Program: 7 **Columnar Transposition**

Date:

**AIM**

**ALGORITHM**

**CODE**

#include <cmath>

#include <iostream>

using namespace std;

int getMinIndex(int arr[], int length) {

    int minValue = INT\_MAX;

    int minIndex = -1;

    for (int i=0; i<length; i++) {

        if (arr[i] < minValue) {

            minIndex = i;

            minValue = arr[i];

        }

    }

    return minIndex;

}

string columnarTranspositionEncryption(string key, string plaintext) {

    int l = plaintext.length();

    int colCount = key.length();

    int rowCount = ceil((float)l / (float) colCount);

    int keyIndices[colCount];

    for (int i=0; i<colCount; i++) {

        keyIndices[i] = (int) ((char) key[i]);

    }

    string cipher;

    // Initialize matrix for the encryption

    char \*\*matrix;

    matrix = new char\*[rowCount];

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

        matrix[i] = new char[colCount];

    }

    // Fill the matrix with plaintext chars row by row

    int count = 0;

    for (int i=0; i<rowCount; i++) {

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

            if (count >= l)

                matrix[i][j] = ' ';

            else

                matrix[i][j] = (char) plaintext[count];

            count++;

        }

    }

    // Pipe the values to result column by column (in ascending order of key chars)

    for (int i=0; i<colCount; i++) {

        int minIndex = getMinIndex(keyIndices, colCount);

        keyIndices[minIndex] = INT\_MAX;

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

            cipher.push\_back(matrix[j][minIndex]);

        }

    }

    return cipher;

}

string columnarTranspositionDecryption(string key, string cipher) {

    int l = cipher.length();

    int colCount = key.length();

    int rowCount = ceil((float)l / (float) colCount);

    int keyIndices[colCount];

    for (int i=0; i<colCount; i++) {

        keyIndices[i] = (int) ((char) key[i]);

    }

    string plaintext;

    // Initialize matrix for the encryption

    char \*\*matrix;

    matrix = new char\*[rowCount];

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

        matrix[i] = new char[colCount];

    }

    // Fill the matrix column by column (in ascending order of key chars)

    int count = 0;

    for (int i=0; i<colCount; i++) {

        int minIndex = getMinIndex(keyIndices, colCount);

        keyIndices[minIndex] = INT\_MAX;

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

            matrix[j][minIndex] = (char) cipher[count];

            count++;

        }

    }

    // Pipe the values to result row by row

    for (int i=0; i<rowCount; i++) {

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

            plaintext.push\_back(matrix[i][j]);

        }

    }

    return plaintext;

}

int main() {

    int choice;

    string key;

    cout << "\nEnter the key: ";

    cin >> key;

    while (1) {

        cout << "\n\n1. Encrypt" << endl;

        cout << "2. Decrypt" << endl;

        cout << "3. Exit" << endl;

        cout << "Enter Choice: ";

        cin >> choice;

        string text;

        if (choice == 1) {

            cout << "\nEnter plaintext: ";

            std::getline(std::cin >> std::ws, text);

            cout << "'" << columnarTranspositionEncryption(key, text) << "'";

        } else if (choice == 2) {

            cout << "\nEnter cipher: ";

            std::getline(std::cin >> std::ws, text);

            cout << "'" << columnarTranspositionDecryption(key, text) << "'";

        } else if (choice == 3) {

            cout << "Exiting.." << endl;

            break;

        } else {

            cout << "Invalid Choice" << endl;

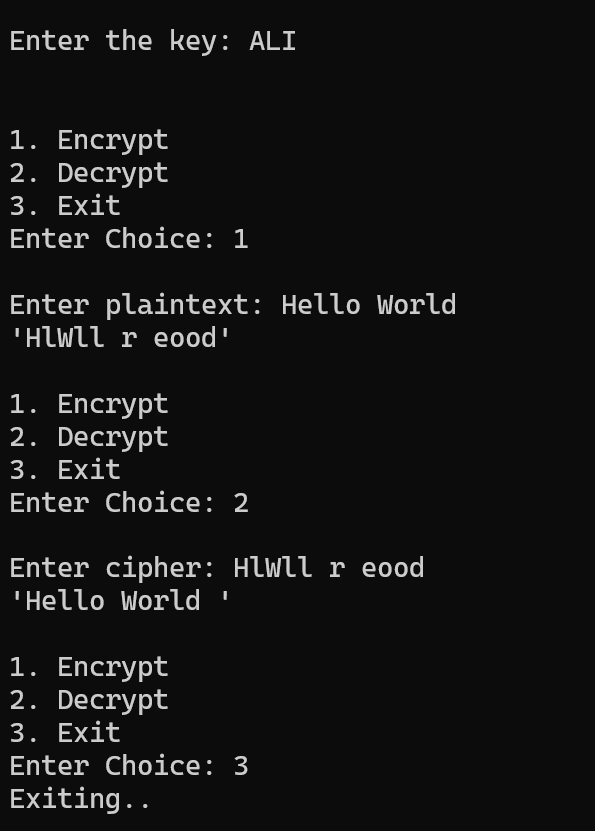
        }

    }

    return 0;

}

**OUTPUT**

****

**RESULT**

Thus, the program to implement encryption and decryption using columnar transposition cipher.