**Information and Network Security**

**2CSDE54**

**Practical 3**

**21BCE020**

**Columnar Transposition Cipher**

#include <bits/stdc++.h>

using *namespace* std;

string encrypt(*char msg[], int key*)

{

*int* count = 0;

    for (*int* i = 0; *msg*[i]; i++)

        if (*msg*[i] != ' ')

*msg*[count++] = *msg*[i];

*msg*[count] = '\0';

*int* msg\_len = strlen(*msg*);

*char* CT\_matrix[*key*][*key*];

    for (*int* i = 0; i < *key*; i++)

    {

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

        {

            CT\_matrix[i][j] = '\*';

        }

    }

*int* k = 0;

    for (*int* i = 0; i < *key*; i++)

    {

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

        {

            if (k < msg\_len)

            {

                CT\_matrix[i][j] = *msg*[k];

                k++;

            }

            else

            {

                CT\_matrix[i][j] = '\*';

            }

        }

    }

    for (*int* i = 0; i < *key*; i++)

    {

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

        {

            if (CT\_matrix[i][j] == '\0')

                break;

            cout << CT\_matrix[i][j] << " ";

        }

        cout << endl;

    }

    string cipher\_text = "";

    cout << "CT: ";

    for (*int* i = 0; i < *key*; i++)

    {

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

        {

            if (CT\_matrix[j][i] != '\*')

            {

                cipher\_text = cipher\_text + CT\_matrix[j][i];

                cout << CT\_matrix[j][i] << "";

            }

        }

    }

    fstream fp;

    fp.open("rail\_op.txt", fstream::out);

    for (*int* i = 0; i < cipher\_text.length(); i++){fp << cipher\_text[i];}

    return cipher\_text;

}

*void* decrypt(*string msg, int key*)

{

*int* count = 0;

    for (*int* i = 0; *msg*[i]; i++)

        if (*msg*[i] != ' ')

*msg*[count++] = *msg*[i];

*msg*[count] = '\0';

*char* dex[*key*][*key*];

    for (*int* i = 0; i < *key*; i++)

    {

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

        {

            dex[i][j] = '\*';

        }

    }

*int* msg\_len = *msg*.length();

    cout << "\n";

*int* k = 0;

*int* x = msg\_len % *key*;

    for (*int* i = 0; i < *key*; i++)

    {

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

        {

            if (i <= (x - 1) || j <= (*key* - 2))

            {

                dex[j][i] = *msg*[k];

                k++;

            }

            else

            {

                dex[j][i] = '\*';

            }

        }

    }

    for (*int* i = 0; i < *key*; i++)

    {

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

        {

            if (dex[i][j] == '\0')

                break;

            cout << dex[i][j] << " ";

        }

        cout << endl;

    }

    string plain\_text = " ";

    cout << "PT: ";

    for (*int* i = 0; i < *key*; i++)

    {

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

        {

            if (dex[i][j] != '\*')

            {

                plain\_text = plain\_text + dex[j][i];

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

            }

        }

    }

}

*int* main()

{

*char* str[100000];

*int* no\_of\_chars = 0;

*int* i = 0;

    ifstream myfile("rail\_ip.txt");

    while (!myfile.eof())

    {

        myfile.get(str[i]);

        i++;

        no\_of\_chars++;

    }

    string op\_str;

*int* l;

    cout << "Length of key: ";

    cin >> l;

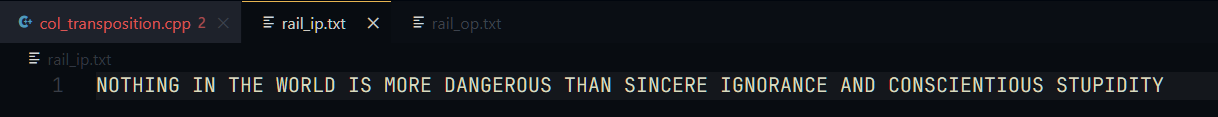
    op\_str = encrypt(str, l);

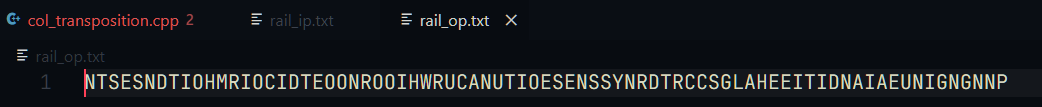
    decrypt(op\_str, l);

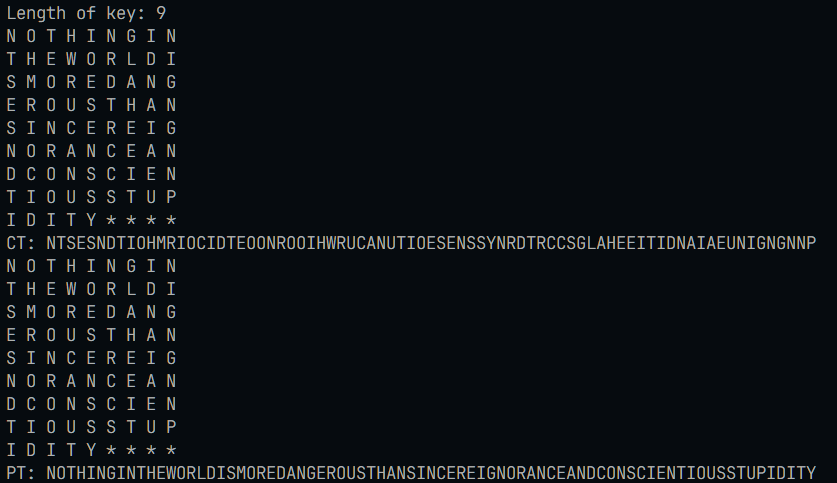
    return 0;

}

**O/P**

****

****



**Keyword Columnar Transposition**

#include <iostream>

#include <fstream>

#include <map>

using *namespace* std;

string const key = "HACK";

map<*int*, *int*> key\_map;

*void* permutation\_order() {for (*int* i = 0; i < key.length(); i++){key\_map[key[i]] = i;}}

string read\_data(*const string &filename*) {

    ifstream file(*filename*);

    string content;

    getline(file, content);

    file.close();

    return content;

}

*void* write\_data(*const string &filename, const string &content*) {

    ofstream file(*filename*);

    file << *content*;

    file.close();

}

string encryptMessage(*string msg*) {

*int* row, col, j;

    string cipher = "";

*//calc col length*

    col = key.length();

*//calc max row 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') {

*//padding to char*

                matrix[i][j] = '\_';

                j++;

            }

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

*//only space and char to mat*

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

                j++;

            }

            k++;

        }

    }

    for (map<*int*, *int*>::iterator ii = key\_map.begin(); ii != key\_map.end(); ++ii)

    {

        j = ii->second;

*//CT from col text using key*

        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(*string cipher*) {

*int* col = key.length(); *//col length*

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

*char* CT\_matrix[row][col];

*//add char to mat*

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

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

            CT\_matrix[i][j] = *cipher*[k++];

*//update order of key for dec*

*int* index = 0;

    for (map<*int*, *int*>::iterator ii = key\_map.begin(); ii != key\_map.end(); ++ii)

        ii->second = index++;

*//arrange matrix col-wise to permutation order*

*char* decCipher[row][col];

    map<*int*, *int*>::iterator ii = key\_map.begin();

*int* k = 0;

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

        j = key\_map[key[l++]];

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

            decCipher[i][k] = CT\_matrix[i][j];

        }

    }

    string msg = ""; *//msg from matrix*

    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(*void*) {

    string msgFile = "keyword\_inp.txt";

    string keyFile = "keyword\_key.txt";

    string msg = read\_data(msgFile);

    string key = read\_data(keyFile);

    permutation\_order();

    string cipher = encryptMessage(msg);

    write\_data("keyword\_enc\_op.txt", cipher);

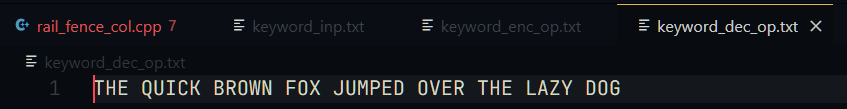
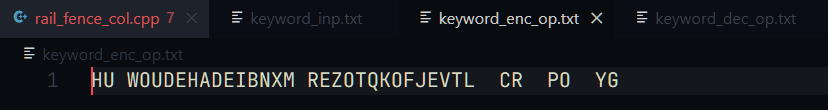
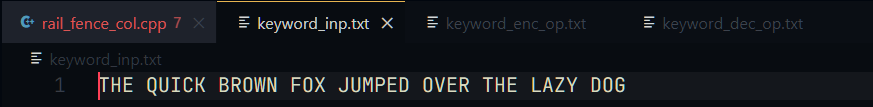
    string decryptedMsg = decryptMessage(cipher);

    write\_data("keyword\_dec\_op.txt", decryptedMsg);

    return 0;

}

**O/P**

****

**ONE PAD CIPHER**

#include <bits/stdc++.h>

using *namespace* std;

string encrypt(*string text, string key*)

{

    string CT = "";

*int* cipher[*key*.length()];

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

    {

        cipher[i] = *text*.at(i) - 'A' + *key*.at(i) - 'A';

        if (cipher[i] > 25)

        {

            cipher[i] = cipher[i] - 26;

        }

    }

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

    {

*int* x = cipher[i] + 'A';

        CT += (*char*)x;

    }

    return CT;

}

static string decrypt(*string s, string key*)

{

    string PT = "";

*int* plain[*key*.length()];

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

    {

        plain[i] = *s*.at(i) - 'A' - (*key*.at(i) - 'A');

        if (plain[i] < 0)

        {

            plain[i] = plain[i] + 26;

        }

    }

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

    {

*int* x = plain[i] + 'A';

        PT += (*char*)x;

    }

    return PT;

}

*int* main()

{

    string PT;

    ifstream inputFile("pad\_ip.txt");

    if (inputFile.is\_open())

    {

        getline(inputFile, PT);

        inputFile.close();

    }

    else

    {

        cout << "Error";

        return 1;

    }

    string key;

    ifstream keyFile("key.txt");

    if (keyFile.is\_open())

    {

        getline(keyFile, key);

        keyFile.close();

    }

    else

    {

        cout << "Error";

        return 1;

    }

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

    {

        PT[i] = toupper(PT[i]);

    }

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

    {

        key[i] = toupper(key[i]);

    }

    string encryptedText = encrypt(PT, key);

    ofstream outputFile("pad\_op.txt");

    if (outputFile.is\_open())

    {

        outputFile << "Cipher text: " << encryptedText << endl;

        outputFile << "Original message: " << decrypt(encryptedText, key);

        outputFile.close();

    }

    else

    {

        cout << "Error";

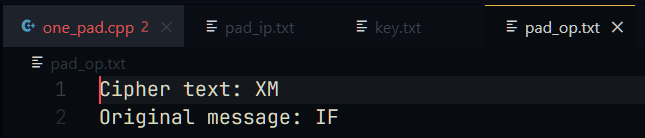
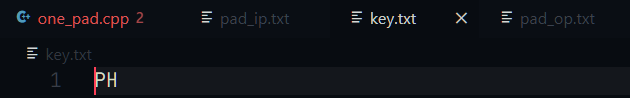
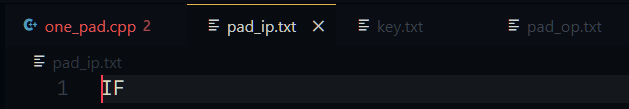
        return 1;

    }

    return 0;

}

**O/P**

****