

**AHSANULLAH UNIVERSITY OF SCIENCE AND TECHNOLOGY**

Department of Computer Science and Engineering

Program: Bachelor of Science in Computer Science and Engineering

Course Code: CSE 4174

Course Title: Cyber Security Lab

Academic Semester: Spring 2023

Assignment Topic: Substitution & Transposition Ciphers

Submitted on: 16/11/2023

Submitted by

Name: Sowpnil Roy

Student ID: 20200104071

Lab Section: B1

1. Devise a code for implementation of Monoalphabetic cipher.

#include <bits/stdc++.h>

**using** **namespace** std**;**

// Function to generate a random cipher key

map**<**char**,** char**>** generateCipherKey**()** **{**

string naturalAlphabet **=** "abcdefghijklmnopqrstuvwxyz"**;**//naturalAlphabet

string changedAlphabet **=** naturalAlphabet**;**//changedAlphabet

random\_device rd**;**

mt19937 g**(**rd**());**

shuffle**(**changedAlphabet**.**begin**(),** changedAlphabet**.**end**(),** g**);**

map**<**char**,** char**>** cipherKey**;**

**for** **(**size\_t i **=** 0**;** i **<** naturalAlphabet**.**length**();** **++**i**)** **{**

cipherKey**[**naturalAlphabet**[**i**]]** **=** changedAlphabet**[**i**];**

**}**

**return** cipherKey**;**

**}**

// Function to encrypt a message using the cipher key

string encryptMessage**(**const string**&** message**,** const map**<**char**,** char**>&** cipherKey**)** **{**

string encryptedMessage**;**

**for** **(**char c **:** message**)** **{**

**if** **(**isalpha**(**c**))** **{**

char encryptedChar **=** cipherKey**.**at**(**tolower**(**c**));**

**if** **(**isupper**(**c**))** **{**

encryptedChar **=** toupper**(**encryptedChar**);**

**}**

encryptedMessage **+=** encryptedChar**;**

**}** **else** **{**

encryptedMessage **+=** c**;**

**}**

**}**

**return** encryptedMessage**;**

**}**

// Function to decrypt an encrypted message using the cipher key

string decryptMessage**(**const string**&** encryptedMessage**,** const map**<**char**,** char**>&** cipherKey**)** **{**

map**<**char**,** char**>** decryptionKey**;**

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

decryptionKey**[**pair**.**second**]** **=** pair**.**first**;**

**}**

string decryptedMessage**;**

**for** **(**char c **:** encryptedMessage**)** **{**

**if** **(**isalpha**(**c**))** **{**

char decryptedChar **=** decryptionKey**.**at**(**tolower**(**c**));**

**if** **(**isupper**(**c**))** **{**

decryptedChar **=** toupper**(**decryptedChar**);**

**}**

decryptedMessage **+=** decryptedChar**;**

**}** **else** **{**

decryptedMessage **+=** c**;**

**}**

**}**

**return** decryptedMessage**;**

**}**

int main**()** **{**

string inputMessage**;**

cout **<<** "Enter a message: "**;**

getline**(**cin**,** inputMessage**);**

map**<**char**,** char**>** cipherKey **=** generateCipherKey**();**

string encryptedMessage **=** encryptMessage**(**inputMessage**,** cipherKey**);**

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

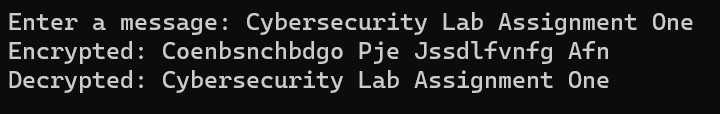
string decryptedMessage **=** decryptMessage**(**encryptedMessage**,** cipherKey**);**

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

**return** 0**;**

**}**

**Input & Output:**



1. Devise a code for implementation of Polyalphabetic cipher.

#include <iostream>

#include <string>

#include <vector>

#include <algorithm>

**using** **namespace** std**;**

string generateKey**(**string key**,** int size**)** **{**

int keyLength **=** key**.**length**();**

string generatedKey**;**

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

generatedKey **+=** key**[**i **%** keyLength**];**

**}**

**return** generatedKey**;**

**}**

string performEncryption**(**string plaintext**,** string key**)** **{**

string ciphertext**;**

int textLength **=** plaintext**.**length**();**

key **=** generateKey**(**key**,** textLength**);**

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

char ch **=** plaintext**[**i**];**

ciphertext **+=** **(**ch **+** key**[**i**])** **%** 128**;**

**}**

**return** ciphertext**;**

**}**

string performDecryption**(**string ciphertext**,** string key**)** **{**

string decryptedText**;**

int textLength **=** ciphertext**.**length**();**

key **=** generateKey**(**key**,** textLength**);**

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

char ch **=** ciphertext**[**i**];**

decryptedText **+=** **(**ch **-** key**[**i**]** **+** 128**)** **%** 128**;**

**}**

**return** decryptedText**;**

**}**

int main**()** **{**

#ifndef ONLINE\_JUDGE

// For getting input from input.txt file

// freopen("input.txt", "r", stdin);

//

// // Printing the Output to output.txt file

// freopen("output.txt", "w", stdout);

//

#endif

string userKey**,** userMessage**;**

cout **<<** "Enter the key: "**;**

getline**(**cin**,** userKey**);**

cout **<<** "Enter the message: "**;**

getline**(**cin**,** userMessage**);**

string encryptedMessage **=** performEncryption**(**userMessage**,** userKey**);**

string decryptedMessage **=** performDecryption**(**encryptedMessage**,** userKey**);**

cout **<<** "Original Message: " **<<** userMessage **<<** endl**;**

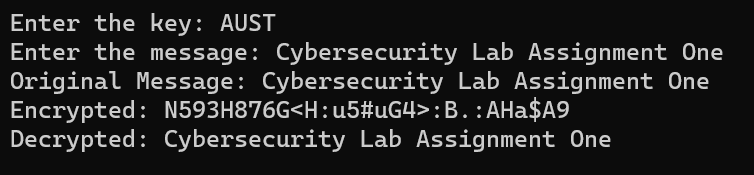
cout **<<** "Encrypted: " **<<** encryptedMessage **<<** endl**;**

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

**return** 0**;**

**}**

**Input & Output:**



1. Devise a code for implementation of Row Transposition cipher.

#include <iostream>

#include <string>

#include <vector>

std**::**string encrypt**(**const std**::**string**&** plaintext**,** const std**::**string**&** key**)** **{**

int cols **=** key**.**length**();**

int rows **=** **(**plaintext**.**length**()** **+** cols **-** 1**)** **/** cols**;**

// Create a 2D vector to represent the matrix

std**::**vector**<**std**::**vector**<**char**>>** matrix**(**rows**,** std**::**vector**<**char**>(**cols**,** 'x'**));**

// Fill the matrix with the plaintext

int index **=** 0**;**

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

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

**if** **(**index **<** plaintext**.**length**())** **{**

matrix**[**i**][**j**]** **=** plaintext**[**index**++];**

**}**

**}**

**}**

// Encrypt the matrix based on the key

std**::**string encrypted**;**

**for** **(**char c **:** key**)** **{**

int col **=** c **-** '0' **-** 1**;**

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

**if** **(**isalpha**(**matrix**[**i**][**col**])** **||** matrix**[**i**][**col**]** **==** ' '**)** **{**

encrypted **+=** matrix**[**i**][**col**];**

**}**

**}**

**}**

**return** encrypted**;**

**}**

std**::**string decrypt**(**const std**::**string**&** ciphertext**,** const std**::**string**&** key**)** **{**

int cols **=** key**.**length**();**

int rows **=** **(**ciphertext**.**length**()** **+** cols **-** 1**)** **/** cols**;**

// Create a 2D vector to represent the matrix

std**::**vector**<**std**::**vector**<**char**>>** matrix**(**rows**,** std**::**vector**<**char**>(**cols**,** 'x'**));**

// Fill the matrix with the ciphertext based on the key

int index **=** 0**;**

**for** **(**char c **:** key**)** **{**

int col **=** c **-** '0' **-** 1**;**

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

matrix**[**i**][**col**]** **=** ciphertext**[**index**++];**

**}**

**}**

// Decrypt the matrix

std**::**string decrypted**;**

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

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

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

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

**}**

**}**

**}**

**return** decrypted**;**

**}**

int main**()** **{**

std**::**string key **=** "213"**;**

std**::**string plaintext **=** "man is mortal"**;**

// Encrypt the plaintext

std**::**string encrypted **=** encrypt**(**plaintext**,** key**);**

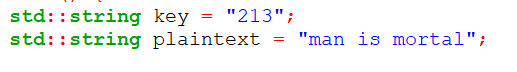
std**::**cout **<<** "Encrypted text: " **<<** encrypted **<<** std**::**endl**;**

// Decrypt the ciphertext

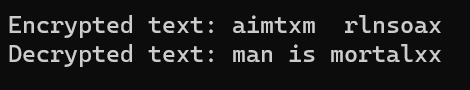
std**::**string decrypted **=** decrypt**(**encrypted**,** key**);**

std**::**cout **<<** "Decrypted text: " **<<** decrypted **<<** std**::**endl**;**

**return** 0**;**

**}**

**Input:**



**Output:**

4. Devise a code for implementation of Column Transposition cipher.

#include <iostream>

#include <string>

#include <vector>

std**::**string encrypt**(**const std**::**string**&** plaintext**,** const std**::**string**&** key**)** **{**

int keyLength **=** key**.**size**();**

std**::**vector**<**std**::**string**>** columns**(**keyLength**,** ""**);**

// Distribute characters of plaintext into columns based on the key

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

columns**[**i **%** keyLength**]** **+=** plaintext**[**i**];**

**}**

// Concatenate the columns to get the ciphertext

std**::**string ciphertext**;**

**for** **(**const std**::**string**&** column **:** columns**)** **{**

ciphertext **+=** column**;**

**}**

**return** ciphertext**;**

**}**

std**::**string decrypt**(**const std**::**string**&** ciphertext**,** const std**::**string**&** key**)** **{**

int keyLength **=** key**.**size**();**

int columnSize **=** ciphertext**.**size**()** **/** keyLength**;**

int extraChars **=** ciphertext**.**size**()** **%** keyLength**;**

std**::**vector**<**std**::**string**>** columns**(**keyLength**,** ""**);**

int index **=** 0**;**

// Distribute characters of ciphertext into columns based on the key

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

int charsInColumn **=** columnSize **+** **(**i **<** extraChars **?** 1 **:** 0**);**

columns**[**i**]** **=** ciphertext**.**substr**(**index**,** charsInColumn**);**

index **+=** charsInColumn**;**

**}**

// Concatenate the columns to get the plaintext

std**::**string plaintext**;**

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

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

**if** **(**i **<** columns**[**j**].**size**())** **{**

plaintext **+=** columns**[**j**][**i**];**

**}**

**}**

**}**

**return** plaintext**;**

**}**

int main**()** **{**

std**::**string plaintext **=** "stpoezjfrckidhifndsw"**;**

std**::**string key **=** "WARE"**;**

std**::**string encrypted **=** encrypt**(**plaintext**,** key**);**

std**::**cout **<<** "Encrypted: " **<<** encrypted **<<** std**::**endl**;**

std**::**string decrypted **=** decrypt**(**encrypted**,** key**);**

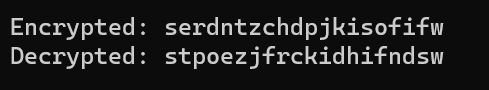
std**::**cout **<<** "Decrypted: " **<<** decrypted **<<** std**::**endl**;**

**return** 0**;**

**}**



**Input:**



**Output:**