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: 20 / 11/ 2023

Submitted by

Name: Parvez Ahammed
Student ID: 20200104129
Lab Section: C2

1. Devise a code for the implementation of Monoalphabetic cipher.

```
#include <bits/stdc++.h>
using namespace std;

map<char, char> monoalphabeticMapping;
string mainString = "qwertyuiopasdfghjklzxcvbnm";
string allChars = "abcdefghijklmnopqrstuvwxyz";
void init()
{

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

        char currentChar = allChars[i];
        char characterMap = mainString[i];

        monoalphabeticMapping[currentChar] = toupper(characterMap);
        monoalphabeticMapping[toupper(currentChar)] = characterMap;
    }
}

string applyMonoAlphabetic(string plainText)
{
    string cipherText = "";

    for (char& ch : plainText) {
        if (isalpha(ch)) {
            char base = isupper(ch) ? 'A' : 'a';
            cipherText += monoalphabeticMapping[ch];
```

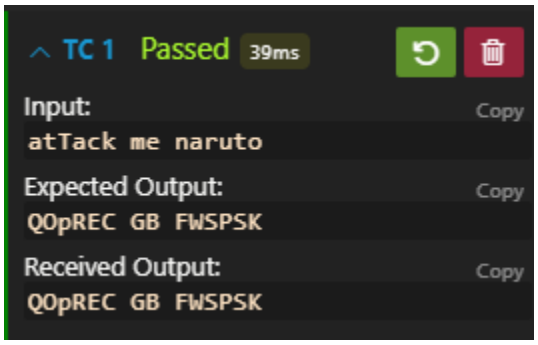
```
        } else
            cipherText += ch;
    }

    return cipherText;
}

int main()
{

    init();
    string plainText, cipher;

    while (cin >> plainText) {

        cipher = applyMonoAlphabetic(plainText);
        cout << cipher << " ";
        cipher.clear();
    }
}

// sample input: attack
// sample output: QZZQEA
```

2. Devise a code for the implementation of Polyalphabetic cipher.

```
#include <bits/stdc++.h>
using namespace std;

map<char, string> polyalphabeticMapping;

string mainString = "qwertyuiopasdfghjklzxcvbnm";
string mainStringUpper = "QWERTYUIOPASDFGHJKLZXCVBNM";
string allChars = "abcdefghijklmnopqrstuvwxyz";




void init()
{
    for (int i = 0; i < allChars.length(); ++i) {
        char currentChar = allChars[i];
        string characterMap = mainString;
        string characterMapUpper = mainStringUpper;

        polyalphabeticMapping[currentChar] = characterMapUpper;
        polyalphabeticMapping[toupper(currentChar)] = characterMap;

        mainString = mainString.back() + mainString.substr(0,
mainString.length() - 1);
        mainStringUpper = mainStringUpper.back() +
mainStringUpper.substr(0, mainStringUpper.length() - 1);
    }

}
```

```
string applyPolyAlphabetic(string plainText)
{
    string cipherText = "";

    int position = 0;

    for (int i = 0; i < plainText.length(); ++i) {
        char currentChar = plainText[i];

        if (currentChar == ' ') {
            cipherText += ' ';
            position = 0;
        } else {
            string mapping = polyalphabeticMapping[currentChar];
            char cipherChar = mapping[position];
            cipherText += cipherChar;
            ++position;
        }
    }

    return cipherText;
}

int main()
{
    init();
    string plainText, cipherText;

    while (cin >> plainText) {
```

```
        cipherText = applyPolyAlphabetic(plainText);
        cout << cipherText << " ";
        cipherText.clear();
    }
}
// sample input: atTack me naruto
// sample output: QOpREC GB FWSPSK
```



3. Devise a code for the implementation of the Row Transposition cipher.

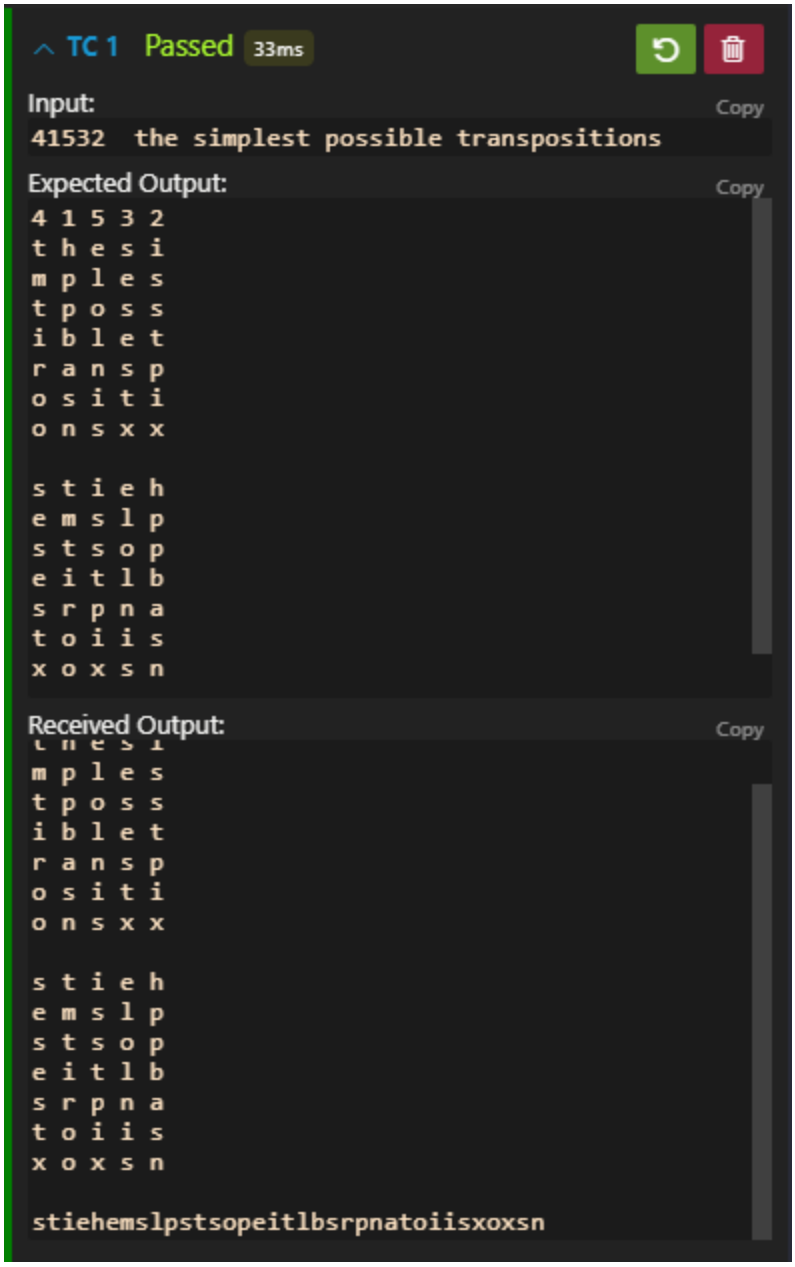
```
#include <bits/stdc++.h>
using namespace std;
vector<string> MATRIX, REARRANGED_MATRIX;
int ROWS = 0;
string KEY, PLAINTEXT = "";
void printMatrix(const vector<string>& MATRIX)
{
    for (string row : MATRIX) {
        for (char ch : row) {
            cout << ch << " ";
        }
        cout << endl;
    }
}
```

```
void fillTheMatrixWithPlainText()
{

    for (int i = 0, k = 0; i < ROWS; ++i) {
        for (int j = 0; j < KEY.length(); ++j) {
            while (k < PLAINTEXT.length() && !isalnum(PLAINTEXT[k]))
{
                k++;
            }

            MATRIX[i][j] = (k < PLAINTEXT.length()) ? PLAINTEXT[k++]
: 'x';
        }
    }
}
void rearrangeTheMatrix()
{

    vector<pair<int, int>> keyWithIndex;
    for (int i = 0; i < KEY.length(); ++i) {
        keyWithIndex.push_back(make_pair(i, KEY[i] - '0'));
    }
    sort(keyWithIndex.begin(), keyWithIndex.end());

    for (int i = 0; i < KEY.length(); i++)
        cout << keyWithIndex[i].second << " ";

    for (int i = 0; i < KEY.length(); i++) {
        for (int j = 0; j < ROWS; j++) {
```

```
            REARRANGED_MATRIX[j][i] =
MATRIX[j][keyWithIndex[i].second - 1];
        }
    }
}
string applyRowTransposition()
{
    ROWS = ceil((double)PLAINTEXT.length() / KEY.length());
    MATRIX.resize(ROWS, string(KEY.length(), ' '));
    REARRANGED_MATRIX.resize(MATRIX.size(), string(KEY.length(), '
'));

    fillTheMatrixWithPlainText();

    rearrangeTheMatrix();

    cout << endl;
    printMatrix(MATRIX);

    cout << endl;
    printMatrix(REARRANGED_MATRIX);

    string cipher;
    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < KEY.length(); j++) {
            cipher += REARRANGED_MATRIX[i][j];
        }
    }

    return cipher;
}
```

```
int main()
{

    cin >> KEY;

    string s;

    while (cin >> s) {
        PLAINTEXT += s;
    }

    string cipherText = applyRowTransposition();
    cout << "\n"
         << cipherText << endl;
}

// Sample input : 41532  the simplest possible transpositions
// Sample Output :

/*
4 1 5 3 2
4 1 5 3 2
t h e s i
m p l e s
t p o s s
i b l e t
r a n s p
o s i t i
o n s x x
```

```
s t i e h
e m s l p
s t s o p
e i t l b
s r p n a
t o i i s
x o x s n

stiehemslpstsopeitlbsrpnatoiisxoxsn
*/
```



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

```
#include <bits/stdc++.h>
using namespace std;
```

```
string KEY, PLAINTEXT;

int ROWS = 0;
vector<string> MATRIX;
vector<pair<int, int>> INDEX;
map<char, int> CHAR_NUMBER;
void printMatrix(const vector<string>& MATRIX)
{
    for (string row : MATRIX) {
        for (char ch : row) {
            cout << ch << " ";
        }
        cout << endl;
    }
}

void fillTheMatrixWithPlainText()
{
    for (int i = 0, k = 0; i < ROWS; ++i) {
        for (int j = 0; j < KEY.length(); ++j) {
            while (k < PLAINTEXT.length() && !isalnum(PLAINTEXT[k]))
{
                k++;
            }

            MATRIX[i][j] = (k < PLAINTEXT.length()) ? PLAINTEXT[k++]
: 'x';
        }
    }
}
```

```
void rearrangeTheMatrix()
{

    string tempkey = KEY;
    sort(tempkey.begin(), tempkey.end());
    for (int i = 0; i < KEY.length(); ++i) {
        CHAR_NUMBER[tempkey[i]] = i + 1;
    }

    vector<pair<int, char>> INDEXED_KEY;
    for (int i = 0; i < KEY.length(); ++i) {
        INDEXED_KEY.push_back({ CHAR_NUMBER[KEY[i]], KEY[i] });
    }

    for (int i = 0; i < KEY.length(); ++i) {
        INDEX.push_back({ CHAR_NUMBER[KEY[i]], i + 1 });
    }

    sort(INDEX.begin(), INDEX.end());
}

string applyColumnarTransposition()
{

    ROWS = ceil((double)(PLAINTEXT.length()) / KEY.length());
    MATRIX.resize(ROWS, string(KEY.length(), ' '));

    fillTheMatrixWithPlainText();
    rearrangeTheMatrix();
```

```
    printMatrix(MATRIX);

    string cipherText;
    for (int i = 0; i < KEY.length(); i++) {
        for (int j = 0; j < ROWS; j++) {
            cipherText += MATRIX[j][INDEX[i].second - 1];
        }
    }

    return cipherText;
}

int main()
{

    cin >> KEY >> PLAINTEXT;

    string s;

    while (cin >> s) {
        PLAINTEXT += s;
    }

    string cipherText = applyColumnarTransposition();
    cout << "\n"
         << cipherText << endl;
}

// Sample Input : HACK meet me after the party
// Sample Output :
```

```
/*
m e e t
m e a f
t e r t
h e p a
r t y x

eeeetearpymmthrtftax
*/
```