Assignment 6

Name :- Aditya Bhusawale
PRN No:- 123B1B094

1. Aim

Write a program to implement the Playfair Cipher and write a conclusion on the strength of the Playfair Cipher.

2. Objective

- To understand the theoretical principles of the Playfair cipher, including key matrix generation and encryption/decryption rules.
- To implement the Playfair cipher algorithm in a programming language.
- To encrypt a given plaintext and decrypt the resulting ciphertext to verify the implementation.
- To analyze the cryptographic strength and weaknesses of the Playfair cipher compared to simpler substitution ciphers.

3. Outcome

Upon completion, we will have a working program that can encrypt and decrypt messages using the Playfair cipher. We will also have a clear understanding of its operation and its relative security in the context of classical cryptography.

4. Theory

The **Playfair Cipher** is a digraph substitution cipher that encrypts pairs of letters instead of single letters. It uses a **5×5 matrix** containing letters of the alphabet (I and J are treated as the same). The matrix is generated using a **keyword**, followed by the remaining unused letters of the alphabet.

Example Keyword:

`MONARCHY`

Key Matrix Generation:

We start by filling the matrix with the unique letters from the keyword **MONARCHY**, then fill the remaining spaces with the rest of the alphabet (excluding 'J', which is merged with 'I'):

M	**O**	**N**	**A**	**R**
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

Encryption Process

Step 1: Prepare the Message

Let's encrypt the message **"INSTRUMENTS".**
Before encryption:

- Replace **J** with **I** (if any).
- Remove spaces and punctuation.

Processed message:
`INSTRUMENTS`

Step 2: Pair the Letters

We divide the message into pairs (digraphs).
If both letters in a pair are the same, insert an **X** between them.
If there's an odd number of letters, append **X** at the end.

`IN ST RU ME NT SX`

Step 3: Apply Encryption Rules

The encryption rules are:

1. **Same Row:** Replace each letter with the one to its right (wrap around if needed).
2. **Same Column:** Replace each letter with the one below it (wrap around if needed).

3. **Different Row and Column:** Form a rectangle and replace each letter with the letter in the same row but in the other letter's column.

Now encrypt each pair:

Pair	Rule Applied	Ciphertext
IN	Rectangle rule → (N, G)	**GS**
ST	Same row → (T, L)	**TL**
RU	Rectangle rule → (M, T)	**MT**
ME	Rectangle rule → (C, K)	**CK**
NT	Rectangle rule → (R, Q)	**RQ**
SX	Rectangle rule → (T, P)	**TP**

Ciphertext:
`GSTLMTCKRQTP`

Decryption Process

Decryption follows the **reverse of encryption rules**:

1. **Same Row:** Replace each letter with the one to its left.
2. **Same Column:** Replace each letter with the one above it.
3. **Different Row and Column:** Form a rectangle and replace each letter with the letter in the same row but in the other letter's column.

Decrypting `GSTLMTCKRQTP` using the key "MONARCHY" gives:

`INSTRUMENTSX` → After removing the padding **X**, we get

INSTRUMENTS

Importance of the Playfair Cipher

During World Wars I and II, the Playfair Cipher was widely used because it was more secure than simple substitution ciphers and required no mechanical devices. However, with the rise of

modern computers, its cryptographic strength declined, as frequency analysis and pattern detection can easily break it today.

Strengths

1. **Digraphic Substitution:** Encrypting pairs of letters (digraphs) hides single-letter frequencies, making it stronger than monoalphabetic ciphers.

2. **Increased Complexity:** There are 26×26 = 676 possible digraphs, making frequency analysis more complex.

3. **Diffusion:** Each plaintext letter affects two ciphertext letters, improving security compared to simple substitution.

5. Conclusion

The **Playfair Cipher** was a remarkable improvement over monoalphabetic substitution ciphers, offering reasonable security for its time. It resisted simple frequency analysis due to digraph substitution and provided better diffusion.

However, it is **not secure by modern standards**, as statistical patterns and computational methods can easily reveal the key and plaintext. Still, it remains an important historical cipher and an excellent teaching tool for understanding classical cryptography concepts.

ass.cpp > generateMatrix(string)

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

char matrix[5][5];

void generateMatrix(string key) {
    string temp = "";
    set<char> used;

    for (char c : key) {
        c = toupper(c);
        if (c == 'J') c = 'I';
        if (!used.count(c) && isalpha(c)) {
            temp += c;
            used.insert(c);
        }
    }

    for (char c = 'A'; c <= 'Z'; c++) {
        if (c == 'J') continue;
        if (!used.count(c)) {
            temp += c;
            used.insert(c);
        }
    }

    int k = 0;
    for (int i = 0; i < 5; i++)
        for (int j = 0; j < 5; j++)
            matrix[i][j] = temp[k++];
}

```

```
pair<int,int> findPos(char c) {
    if (c == 'J') c = 'I';
    for (int i = 0; i < 5; i++)
        for (int j = 0; j < 5; j++)
            if (matrix[i][j] == c)
                return {i, j};
    return {-1, -1};
}

string prepareText(string text) {
    string res = "";
    for (char c : text) {
        if (isalpha(c)) res += toupper(c);
    }

    string finalText = "";
    for (int i = 0; i < res.size(); i++) {
        char first = res[i];
        char second = (i + 1 < res.size()) ? res[i+1] : 'X';
        if (first == second) {
            finalText += first;
            finalText += 'X';
        } else {
            finalText += first;
            if (i + 1 < res.size()) {
                finalText += second;
                i++;
            }
        }
    }
    if (finalText.size() % 2 != 0)
        finalText += 'X';
    return finalText;
}
```

```
string encrypt(string plaintext) {
    string cipher = "";
    for (int i = 0; i < plaintext.size(); i += 2) {
        char a = plaintext[i];
        char b = plaintext[i+1];
        auto pos1 = findPos(a);
        auto pos2 = findPos(b);

        if (pos1.first == pos2.first) {
            cipher += matrix[pos1.first][(pos1.second + 1) % 5];
            cipher += matrix[pos2.first][(pos2.second + 1) % 5];
        } else if (pos1.second == pos2.second) {
            cipher += matrix[(pos1.first + 1) % 5][pos1.second];
            cipher += matrix[(pos2.first + 1) % 5][pos2.second];
        } else {
            cipher += matrix[pos1.first][pos2.second];
            cipher += matrix[pos2.first][pos1.second];
        }
    }
    return cipher;
}

int main() {
    string key, plaintext;
    cout << "Enter Key: ";
    getline(cin, key);
    cout << "Enter Plaintext: ";
    getline(cin, plaintext);

    generateMatrix(key);
    string preparedText = prepareText(plaintext);
    string cipherText = encrypt(preparedText);

    cout << "\nPlayfair Cipher Matrix:\n";
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++)
            cout << matrix[i][j] << " ";
        cout << "\n";
    }

    cout << "\nPrepared Text: " << preparedText << endl;
    cout << "Cipher Text: " << cipherText << endl;

    return 0;
}
```

```
Enter Key: MONARCHY
Enter Plaintext: HELLO WORLD

Playfair Cipher Matrix:
M O N A R
C H Y B D
E F G I K
L P Q S T
U V W X Z

Prepared Text: HELXLOWORLDX
Cipher Text: CFSUPMVNMTBZ
```

```
Enter Key: MONARCHY
Enter Plaintext: ADITYA BHUSAWALE

Playfair Cipher Matrix:
M O N A R
C H Y B D
E F G I K
L P Q S T
U V W X Z

Prepared Text: ADITYABHUSAWALEX
Cipher Text: RBKSBNDYXLNXMSIU
```