Implement error correcting code.

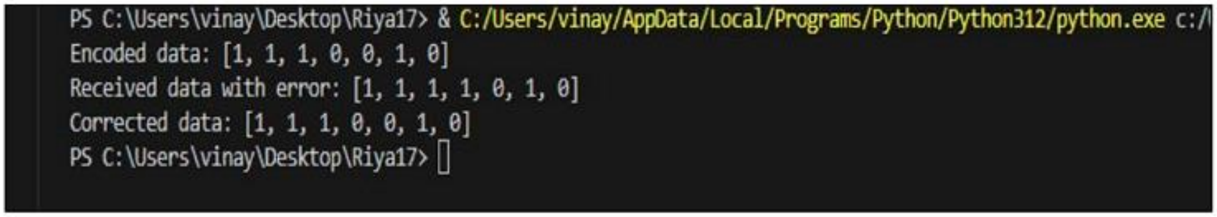
```
def add_parity_bits(data):

"""Add parity bits to the data."""

# Calculate the number of parity bits needed

parity_bits_count = 0

while 2 ** parity_bits_count <= len(data) + parity_bits_count:

parity_bits_count += 1


# Insert parity bits

encoded_data = []

j = 0

for i in range(1, len(data) + parity_bits_count + 1):

if i == 2 ** j:

encoded_data.append(0) # Placeholder for parity bit

j += 1

else:

encoded_data.append(data.pop(0))


# Calculate parity bits

for i in range(parity_bits_count):

parity_index = 2 ** i - 1

parity = 0

for j in range(parity_index, len(encoded_data), 2 * (parity_index + 1)):

parity ^= encoded_data[j]

encoded_data[parity_index] = parity


return encoded_data


def correct_errors(encoded_data):

"""Correct errors in the encoded data."""

parity_bits_count = 0

while 2 ** parity_bits_count <= len(encoded_data):

parity_bits_count += 1


# Calculate the syndrome

syndrome = 0

for i in range(parity_bits_count):

parity_index = 2 ** i - 1

parity = 0

for j in range(parity_index, len(encoded_data), 2 * (parity_index + 1)):

parity ^= encoded_data[j]

syndrome |= parity << i


# Correct errors if any
```

```python
if syndrome != 0:
error_index = syndrome - 1
encoded_data[error_index] ^= 1

return encoded_data

def hamming_encode(data):
"""Encode the data using Hamming (7, 4) code."""

encoded_data = add_parity_bits(data)
return encoded_data

def hamming_decode(encoded_data):
"""Decode the Hamming encoded data."""
corrected_data = correct_errors(encoded_data)
return corrected_data

# Example usage:
data = [1, 0, 1, 0] # 4-bit data
encoded_data = hamming_encode(data)
print("Encoded data:", encoded_data)

# Simulating an error in transmission
encoded_data[3] = 1 - encoded_data[3] # Introducing an error
print("Received data with error:", encoded_data)

corrected_data = hamming_decode(encoded_data)
print("Corrected data:", corrected_data)
```

```cpp
#include <iostream>
#include <vector>

using namespace std;

// Function to calculate the parity bit
int calculateParityBit(const vector<int>& data, int position) {
    int parity = 0;
    for (size_t i = 0; i < data.size(); ++i) {
        if ((i + 1) & (1 << position)) {
            parity ^= data[i];
        }
    }
    return parity;
}

// Function to encode the data using Hamming(7,4) code
vector<int> hammingEncode(const vector<int>& data) {
    vector<int> encodedData(7);
    encodedData[2] = data[0];
    encodedData[4] = data[1];
    encodedData[5] = data[2];
    encodedData[6] = data[3];

    encodedData[0] = calculateParityBit(encodedData, 0);
    encodedData[1] = calculateParityBit(encodedData, 1);
    encodedData[3] = calculateParityBit(encodedData, 3);

    return encodedData;
}

// Function to correct the encoded data
void correctError(vector<int>& encodedData) {
    int errorPosition = 0;
    for (int i = 0; i < 3; ++i) {
        errorPosition |= (calculateParityBit(encodedData, i) << i);
    }

    if (errorPosition != 0) {
        cout << "Error detected at position " << errorPosition << ". Correcting...\n";
        encodedData[errorPosition - 1] ^= 1;
    } else {
        cout << "No error detected.\n";
    }
}

int main() {
    // Example usage
    vector<int> data = {1, 0, 1, 0}; // Data bits
    vector<int> encodedData = hammingEncode(data);

    cout << "Encoded data: ";
    for (int bit : encodedData) {
        cout << bit;
    }
    cout << endl;

    // Simulate an error
    encodedData[1] ^= 1;

    // Correct the error
    correctError(encodedData);

    cout << "Corrected data: ";
    for (int bit : encodedData) {
        cout << bit;
    }
    cout << endl;
```

```
   return 0;

}
```

OUTPUT:



```
PS C:\Users\vinay\Desktop\Riya17> & C:/Users/vinay/AppData/Local/Programs/Python/Python312/python.exe c:/
Encoded data: [1, 1, 1, 0, 0, 1, 0]
Received data with error: [1, 1, 1, 1, 0, 1, 0]
Corrected data: [1, 1, 1, 0, 0, 1, 0]
PS C:\Users\vinay\Desktop\Riya17> 
```

2.

```
#include <iostream>

#include <bitset>


// Function to calculate the parity bit for a given data word

int calculateParityBit(int data) {

int parity = 0;


// Calculate parity by XORing all the bits

while (data) {

parity ^= (data & 1);

data >>= 1;

}


return parity;

}


// Function to encode the data with a parity bit

int addParityBit(int data) {

// Calculate the parity bit

int parity = calculateParityBit(data);
// Add the parity bit to the least significant bit position
return (data << 1) | parity;
}
```

Implement the error detecting code cpp

```
/ Function to check if there is any error in the received data
bool checkError(int receivedData) {
// Calculate the parity bit of the received data
int receivedParity = calculateParityBit(receivedData);

// Extract the received parity bit
int receivedBit = receivedData & 1;

// If the calculated parity and received parity don't match,
there is an error
return (receivedBit != receivedParity);
}

int main() {
// Example usage
int data = 0b1011; // Data to be sent
int encodedData = addParityBit(data); // Add parity bit
std::cout << "Encoded Data: " << std::bitset<5>(encodedData)
<< std::endl;

// Simulate error by flipping a bit
int receivedData = encodedData ^ (1 << 2); // Flipping the
third bit
// Check for error
bool errorDetected = checkError(receivedData);

if (errorDetected) {
std::cout << "Error detected in received data!" << std::endl;
} else {
std::cout << "No error detected in received data." << std::endl;
}

return 0;
}
```

Practical 3

Implement caesar cipher substitution operation

Code:

```
#include <iostream>
using namespace std;

// This function receives text and shift and
// returns the encrypted text
string encrypt(string text, int s)
{
    string result = "";

    // traverse text
    for (int i = 0; i < text.length(); i++) {
        // apply transformation to each character
        // Encrypt Uppercase letters
        if (isupper(text[i]))
            result += char(int(text[i] + s - 65) % 26 + 65);

        // Encrypt Lowercase letters
        else
            result += char(int(text[i] + s - 97) % 26 + 97);
    }

    // Return the resulting string
    return result;
}

// Driver program to test the above function
int main()
{
    string text = "ATTACKATONCE";
    int s = 4;
    cout << "Text : " << text;
    cout << "\nShift: " << s;
    cout << "\nCipher: " << encrypt(text, s);
    return 0;
}
```

Implement monoalphabetic and polyalphabetic cipher substitution.

Code:

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
```

```
using namespace std;

// Function to encrypt a message using monoalphabetic substitution cipher
string monoalphabeticEncrypt(const string& message, const string& key) {
string encryptedMessage = message;
for (char& c : encryptedMessage) {
if (isalpha(c)) {
char base = isupper(c) ? 'A' : 'a';
c = key[c - base];
}
}
return encryptedMessage;
}

// Function to decrypt a message using monoalphabetic substitution cipher
string monoalphabeticDecrypt(const string& encryptedMessage, const string& key) {
string decryptedMessage = encryptedMessage;
for (char& c : decryptedMessage) {
if (isalpha(c)) {
char base = isupper(c) ? 'A' : 'a';
c = 'A' + distance(key.begin(), find(key.begin(), key.end(), c));
}
```

```
}
```

```
return decryptedMessage;
}

// Function to encrypt a message using polyalphabetic substitution cipher (Vigenere cipher)
string polyalphabeticEncrypt(const string& message, const string& key) {
string encryptedMessage;
int keyIndex = 0;
for (char c : message) {
if (isalpha(c)) {
char base = isupper(c) ? 'A' : 'a';
char shifted = ((c - base) + (key[keyIndex % key.length()] - 'A')) % 26 + base;
encryptedMessage.push_back(shifted);
keyIndex++;
} else {
encryptedMessage.push_back(c);
}
}
return encryptedMessage;
}

// Function to decrypt a message using polyalphabetic substitution cipher (Vigenere cipher)
string polyalphabeticDecrypt(const string& encryptedMessage, const string& key) {
string decryptedMessage;
int keyIndex = 0;
for (char c : encryptedMessage) {
if (isalpha(c)) {
char base = isupper(c) ? 'A' : 'a';
char shifted = ((c - base) - (key[keyIndex % key.length()] - 'A') + 26) % 26 + base;
decryptedMessage.push_back(shifted);
keyIndex++;
} else {
decryptedMessage.push_back(c);
}
}
return decryptedMessage;
}

int main() {
string message = "Hello, World!";
string monoalphabeticKey = "ZYXWVUTSRQPONMLKJIHGFEDCBA";
string polyalphabeticKey = "KEY";

// Encrypt using monoalphabetic substitution cipher
string encryptedMonoalphabetic = monoalphabeticEncrypt(message,
monoalphabeticKey);
cout << "Monoalphabetic Encrypted: " << encryptedMonoalphabetic << endl;

// Decrypt using monoalphabetic substitution cipher
string decryptedMonoalphabetic = monoalphabeticDecrypt(encryptedMonoalphabetic,
monoalphabeticKey);
cout << "Monoalphabetic Decrypted: " << decryptedMonoalphabetic << endl;

// Encrypt using polyalphabetic substitution cipher (Vigenere cipher)
string encryptedPolyalphabetic = polyalphabeticEncrypt(message, polyalphabeticKey);
cout << "Polyalphabetic Encrypted: " << encryptedPolyalphabetic << endl;
```

```
// Decrypt using polyalphabetic substitution cipher (Vigenere cipher)
```

5

```
string decryptedPolyalphabetic = polyalphabeticDecrypt(encryptedPolyalphabetic,
polyalphabeticKey);
cout << "Polyalphabetic Decrypted: " << decryptedPolyalphabetic << endl;

return 0;
}
```

Output:

Practical 5

Q5- Implement playfair cipher substitution operation.

Code-

```
def prepare_input(text):

# Remove spaces and convert to uppercase
text = text.replace(" ", "").upper()
# Replace 'J' with 'I'
text = text.replace("J", "I")
# Split the text into pairs of letters
pairs = []
for i in range(0, len(text), 2):
pair = text[i:i+2]

if len(pair) == 1: # If the last pair has only one letter, add 'X' to

make it a pair
pair += 'X'
pairs.append(pair)
return pairs
```

```
def generate_key_matrix(key):
# Remove spaces and convert to uppercase
key = key.replace(" ", "").upper()
# Replace 'J' with 'I'
key = key.replace("J", "I")
# Create a set of unique letters from the key (without duplicates)
key_set = list(dict.fromkeys(key))
# Create the key matrix (5x5 grid)
key_matrix = [['' for _ in range(5)] for _ in range(5)]
i, j = 0, 0
for letter in key_set:
key_matrix[i][j] = letter

j += 1
if j == 5:
j = 0
i += 1
# Fill the remaining spaces with the remaining letters of the alphabet
(excluding 'J')
alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"
for letter in alphabet:
if letter not in key_set:
key_matrix[i][j] = letter
j += 1
if j == 5:
j = 0
i += 1
return key_matrix
def find_letter_positions(letter, key_matrix):
for i in range(5):
for j in range(5):
if key_matrix[i][j] == letter:
return (i, j)
def encrypt(plaintext, key):
pairs = prepare_input(plaintext)
key_matrix = generate_key_matrix(key)
cipher_text = ''
for pair in pairs:
char1, char2 = pair[0], pair[1]
row1, col1 = find_letter_positions(char1, key_matrix)
row2, col2 = find_letter_positions(char2, key_matrix)
if row1 == row2: # Same row
cipher_text += key_matrix[row1][(col1 + 1) % 5]
cipher_text += key_matrix[row2][(col2 + 1) % 5]
elif col1 == col2: # Same column
```

```
cipher_text += key_matrix[(row1 + 1) % 5][col1]
cipher_text += key_matrix[(row2 + 1) % 5][col2]
else: # Forming rectangle
cipher_text += key_matrix[row1][col2]
cipher_text += key_matrix[row2][col1]
return cipher_text
def decrypt(ciphertext, key):
pairs = prepare_input(ciphertext)
key_matrix = generate_key_matrix(key)
plain_text = ''

for pair in pairs:
char1, char2 = pair[0], pair[1]
row1, col1 = find_letter_positions(char1, key_matrix)
row2, col2 = find_letter_positions(char2, key_matrix)
if row1 == row2: # Same row
plain_text += key_matrix[row1][(col1 - 1) % 5]
plain_text += key_matrix[row2][(col2 - 1) % 5]
elif col1 == col2: # Same column
plain_text += key_matrix[(row1 - 1) % 5][col1]
plain_text += key_matrix[(row2 - 1) % 5][col2]
else: # Forming rectangle
plain_text += key_matrix[row1][col2]
plain_text += key_matrix[row2][col1]
return plain_text
def main():
key = input("Enter the key for Playfair cipher: ")
plaintext = input("Enter the plaintext: ")
encrypted_text = encrypt(plaintext, key)
print("Encrypted Text:", encrypted_text)
decrypted_text = decrypt(encrypted_text, key)
print("Decrypted Text:", decrypted_text)
if __name__ == "__main__":
main()
```

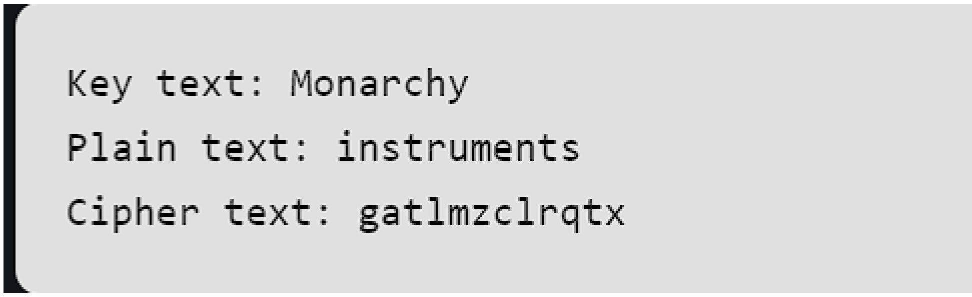
```
#include <bits/stdc++.h>
using namespace std;
#define SIZE 30

// Function to convert the string to lowercase
void toLowerCase(char plain[], int ps)
{
    int i;
    for (i = 0; i < ps; i++) {
        if (plain[i] > 64 && plain[i] < 91)
            plain[i] += 32;
    }
}

// Function to remove all spaces in a string
int removeSpaces(char* plain, int ps)
{
    int i, count = 0;
    for (i = 0; i < ps; i++)
        if (plain[i] != ' ')
            plain[count++] = plain[i];
    plain[count] = '\0';
    return count;
}

// Function to generate the 5x5 key square
void generateKeyTable(char key[], int ks, char keyT[5][5])
{
    int i, j, k, flag = 0;

    // a 26 character hashmap
    // to store count of the alphabet
    int dicty[26] = { 0 };
    for (i = 0; i < ks; i++) {
        if (key[i] != 'j')
            dicty[key[i] - 97] = 2;
    }

    dicty['j' - 97] = 1;

    i = 0;
    j = 0;

    for (k = 0; k < ks; k++) {
        if (dicty[key[k] - 97] == 2) {
            dicty[key[k] - 97] -= 1;
            keyT[i][j] = key[k];
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }

    for (k = 0; k < 26; k++) {
        if (dicty[k] == 0) {
            keyT[i][j] = (char)(k + 97);
            j++;
            if (j == 5) {
                i++;
                j = 0;
            }
        }
    }
```

```
}

// Function to search for the characters of a digraph
// in the key square and return their position
void search(char keyT[5][5], char a, char b, int arr[])
{
    int i, j;

    if (a == 'j')
        a = 'i';
    else if (b == 'j')
        b = 'i';

    for (i = 0; i < 5; i++) {

        for (j = 0; j < 5; j++) {

            if (keyT[i][j] == a) {
                arr[0] = i;
                arr[1] = j;
            }
            else if (keyT[i][j] == b) {
                arr[2] = i;
                arr[3] = j;
            }
        }
    }
}
```

```
// Function to find the modulus with 5
int mod5(int a) { return (a % 5); }

// Function to make the plain text length to be even
int prepare(char str[], int ptrs)
{
    if (ptrs % 2 != 0) {
        str[ptrs++] = 'z';
        str[ptrs] = '\0';
    }
    return ptrs;
}

// Function for performing the encryption
void encrypt(char str[], char keyT[5][5], int ps)
{
    int i, a[4];

    for (i = 0; i < ps; i += 2) {

        search(keyT, str[i], str[i + 1], a);

        if (a[0] == a[2]) {
            str[i] = keyT[a[0]][mod5(a[1] + 1)];
            str[i + 1] = keyT[a[0]][mod5(a[3] + 1)];
        }
        else if (a[1] == a[3]) {
            str[i] = keyT[mod5(a[0] + 1)][a[1]];
            str[i + 1] = keyT[mod5(a[2] + 1)][a[1]];
        }
        else {
            str[i] = keyT[a[0]][a[3]];
            str[i + 1] = keyT[a[2]][a[1]];
        }
    }
}

// Function to encrypt using Playfair Cipher
void encryptByPlayfairCipher(char str[], char key[])
{
    char ps, ks, keyT[5][5];

    // Key
    ks = strlen(key);
    ks = removeSpaces(key, ks);
    toLowerCase(key, ks);

    // Plaintext
    ps = strlen(str);
    toLowerCase(str, ps);
    ps = removeSpaces(str, ps);

    ps = prepare(str, ps);

    generateKeyTable(key, ks, keyT);

    encrypt(str, keyT, ps);
}
```

```
// Driver code
int main()
{
    char str[SIZE], key[SIZE];

    // Key to be encrypted
    strcpy(key, "Monarchy");
    cout << "Key text: " << key << "\n";

    // Plaintext to be encrypted
    strcpy(str, "instruments");
    cout << "Plain text: " << str << "\n";

    // encrypt using Playfair Cipher
    encryptByPlayfairCipher(str, key);

    cout << "Cipher text: " << str << "\n";

    return 0;
}
```

Output-



```
Key text: Monarchy
Plain text: instruments
Cipher text: gatlmzclrqtx
```

2

Implement hill cipher substitution operation.

Code:

```
#include<iostream>
#include<math.h>
using namespace std;
float en[3][1], de[3][1], a[3][3], b[3][3], msg[3][1], m[3][3];
void getKeyMatrix() { //get key and message from user
int i, j;
char mes[3];
cout<<"Enter 3x3 matrix for key (should have inverse):\n";
for(i = 0; i < 3; i++)
for(j = 0; j < 3; j++) {
cin>>a[i][j];
m[i][j] = a[i][j];
}
cout<<"\nEnter a string of 3 letter(use A through Z): ";
cin>>mes;
for(i = 0; i < 3; i++)
msg[i][0] = mes[i] - 65;
}
void encrypt() { //encrypts the message
int i, j, k;
for(i = 0; i < 3; i++)
for(j = 0; j < 1; j++)
for(k = 0; k < 3; k++)
en[i][j] = en[i][j] + a[i][k] * msg[k][j];
cout<<"\nEncrypted string is: ";
```

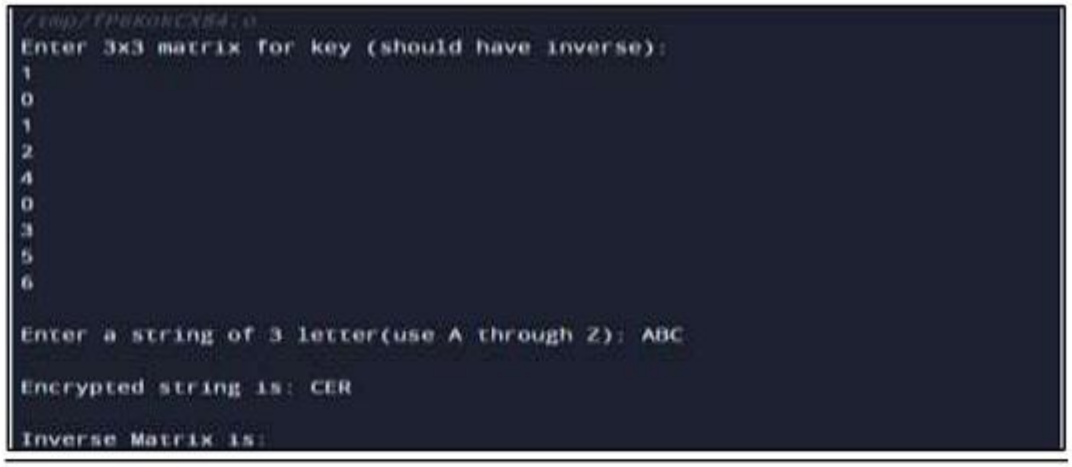
```
for(i = 0; i < 3; i++)
```

```
cout<<(char)(fmod(en[i][0], 26) + 65); //modulo 26 is taken for each element of the matrix
obtained by multiplication
}
void inversematrix() { //find inverse of key matrix
int i, j, k;
float p, q;
for(i = 0; i < 3; i++)
for(j = 0; j < 3; j++) {
if(i == j)
b[i][j]=1;
else
b[i][j]=0;
}
for(k = 0; k < 3; k++) {
for(i = 0; i < 3; i++) {
p = m[i][k];
q = m[k][k];
for(j = 0; j < 3; j++) {
if(i != k) {
m[i][j] = m[i][j]*q - p*m[k][j];
b[i][j] = b[i][j]*q - p*b[k][j];
}
}
}
}
for(i = 0; i < 3; i++)
for(j = 0; j < 3; j++)
b[i][j] = b[i][j] / m[i][i];
cout<<"\n\nInverse Matrix is:\n";
for(i = 0; i < 3; i++) {
```

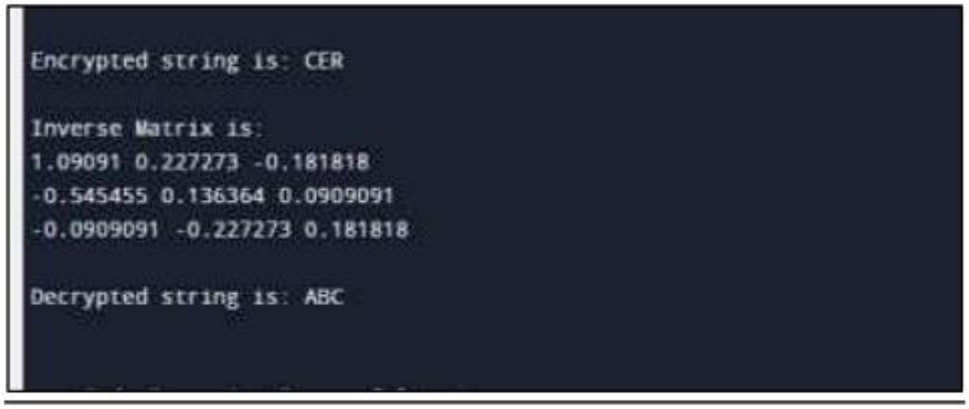
```
for(j = 0; j < 3; j++)
cout<<b[i][j]<<" ";
cout<<"\n";
}
}
void decrypt() { //decrypt the message
int i, j, k;
inversematrix();
for(i = 0; i < 3; i++)
for(j = 0; j < 1; j++)
for(k = 0; k < 3; k++)
de[i][j] = de[i][j] + b[i][k] * en[k][j];
cout<<"\nDecrypted string is: ";
for(i = 0; i < 3; i++)
cout<<(char)(fmod(de[i][0], 26) + 65); //modulo 26 is taken to get the original message
cout<<"\n";
}
int main() {
getKeyMatrix();
```

```
encrypt();

decrypt();

}
```

Output:



```
/tmp/fP6K0KCXB4.o
Enter 3x3 matrix for key (should have inverse):
1
0
1
2
4
0
3
5
6

Enter a string of 3 letter(use A through Z): ABC

Encrypted string is: CER

Inverse Matrix is:
```



```
Encrypted string is: CER

Inverse Matrix is:
1.09091 0.227273 -0.181818
-0.545455 0.136364 0.0909091
-0.0909091 -0.227273 0.181818

Decrypted string is: ABC
```

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

string railFenceEncrypt(const string& plaintext, int rails) {
vector<string> fence(rails);
int row = 0;
bool down = false;

for (char c : plaintext) {
fence[row] += c;
if (row == 0 || row == rails - 1) {
down = !down;
}
row += down ? 1 : -1;
}

string ciphertext;
for (const string& rail : fence) {
ciphertext += rail;
}

return ciphertext;
}
```

Practical 7

Implement rail fence cipher transposition operation.

3

```
string railFenceDecrypt(const string& ciphertext, int rails) {
vector<string> fence(rails);
vector<int> indices(ciphertext.size());

int row = 0;
bool down = false;

for (int i = 0; i < ciphertext.size(); ++i) {
indices[i] = row;
if (row == 0 || row == rails - 1) {
down = !down;
}
row += down ? 1 : -1;
}

int index = 0;
for (int i = 0; i < rails; ++i) {
for (int j = 0; j < indices.size(); ++j) {
if (indices[j] == i) {
fence[i] += ciphertext[index++];
}
}
}

string plaintext;
```

```
row = 0;
down = false;

for (int i = 0; i < ciphertext.size(); ++i) {
plaintext += fence[row][0];

fence[row].erase(0, 1);
if (row == 0 || row == rails - 1) {
down = !down;
}
row += down ? 1 : -1;
}

return plaintext;
}

int main() {
string plaintext, encrypted, decrypted;
int rails;

cout << "Enter the plaintext: ";
getline(cin, plaintext);

cout << "Enter the number of rails: ";
cin >> rails;

encrypted = railFenceEncrypt(plaintext, rails);
cout << "Encrypted: " << encrypted << endl;
```

```
decrypted = railFenceDecrypt(encrypted, rails);
cout << "Decrypted: " << decrypted << endl;

return 0;
}
```

4

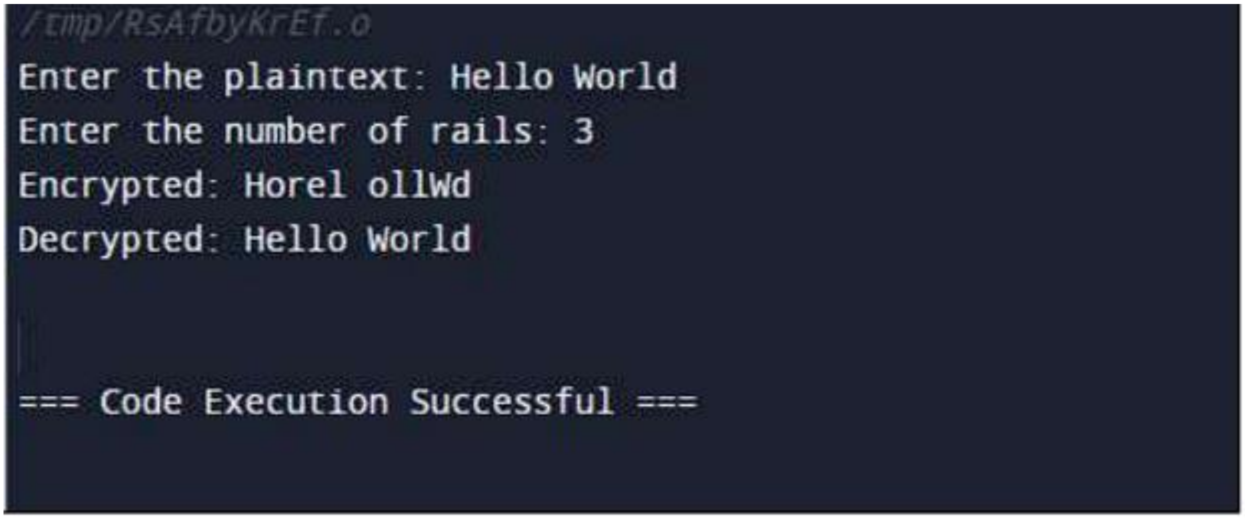
5

OUTPUT:

2

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

string rowTranspositionEncrypt(const string& plaintext, const vector<int>& key) {
int rows = key.size();
int cols = (plaintext.size() + rows - 1) / rows;

vector<vector<char>> matrix(rows, vector<char>(cols, ' '));
int index = 0;

for (int col = 0; col < cols; col++) {
```

```
for (int row : key) {
if (index < plaintext.size()) {
matrix[row][col] = plaintext[index++];
} else {
break;
}
}
}

string ciphertext;
for (int row = 0; row < rows; row++) {
for (int col = 0; col < cols; col++) {
```



```
/tmp/RsAfbyKrEf.o
Enter the plaintext: Hello World
Enter the number of rails: 3
Encrypted: Horel ollWd
Decrypted: Hello World


=== Code Execution Successful ===
```

Practical 8

Implement row transposition cipher transposition operation.

3

```
ciphertext += matrix[row][col];
}
```

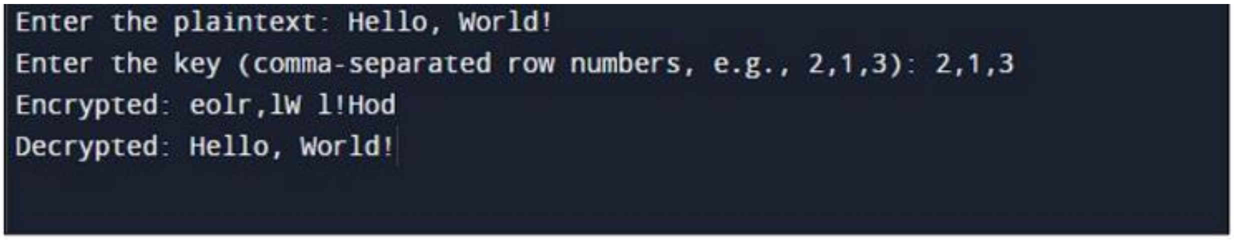
```
}

return ciphertext;
}

string rowTranspositionDecrypt(const string& ciphertext, const vector<int>& key) {
int rows = key.size();
int cols = (ciphertext.size() + rows - 1) / rows;

vector<vector<char>> matrix(rows, vector<char>(cols, ' '));
int index = 0;

for (int row = 0; row < rows; row++) {
for (int col = 0; col < cols; col++) {
matrix[row][col] = ciphertext[index++];
}
}

string plaintext;
for (int col = 0; col < cols; col++) {
for (int row : key) {
plaintext += matrix[row][col];
}
}

return plaintext;
}

int main() {
```

4

```
string plaintext, encrypted, decrypted;
vector<int> key;

cout << "Enter the plaintext: ";
getline(cin, plaintext);

cout << "Enter the key (comma-separated row numbers, e.g., 2,1,3): ";
string keyInput;
getline(cin, keyInput);

size_t pos = 0;
while ((pos = keyInput.find(',')) != string::npos) {
key.push_back(stoi(keyInput.substr(0, pos)));
keyInput.erase(0, pos + 1);
}
key.push_back(stoi(keyInput)); // Add the last key element

encrypted = rowTranspositionEncrypt(plaintext, key);
cout << "Encrypted: " << encrypted << endl;

decrypted = rowTranspositionDecrypt(encrypted, key);
cout << "Decrypted: " << decrypted << endl;

return 0;
}
```

5

OUTPUT:



```
Enter the plaintext: Hello, World!
Enter the key (comma-separated row numbers, e.g., 2,1,3): 2,1,3
Encrypted: eolr,lW l!Hod
Decrypted: Hello, World!
```

Practical File

Q9- **Implement product cipher transposition operation.**

Code-

```
def encrypt(text, key):


encrypted_text = [''] * len(key)

# Arrange the text based on the key

for i in range(len(key)):

encrypted_text[key[i] - 1] = text[i]

return ''.join(encrypted_text)

def decrypt(text, key):

decrypted_text = [''] * len(key)

# Rearrange the text based on the key

for i in range(len(key)):

decrypted_text[i] = text[key[i] - 1]

return ''.join(decrypted_text)

def main():

choice = input("Do you want to (e)ncrypt or (d)ecrypt? ").lower()

if choice == 'e':

text = input("Enter the text to encrypt: ")

key = list(map(int, input("Enter the encryption key (sequence of numbers

from 1 to n separated by spaces): ").split()))

if sorted(key) != list(range(1, len(key) + 1)):

print("Invalid key. Key should be a sequence of numbers from 1 to

n.")

return

encrypted_text = encrypt(text, key)

print("Encrypted text:", encrypted_text)
```

```
elif choice == 'd':

text = input("Enter the text to decrypt: ")

key = list(map(int, input("Enter the decryption key (sequence of numbers

from 1 to n separated by spaces): ").split()))

if sorted(key) != list(range(1, len(key) + 1)):

print("Invalid key. Key should be a sequence of numbers from 1 to

n.")

return

decrypted_text = decrypt(text, key)

print("Decrypted text:", decrypted_text)

else:

print("Invalid choice.")

if __name__ == "__main__":

main()
```

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

// Function to perform Caesar cipher encryption
string caesarEncrypt(const string& plaintext, int shift) {
    string ciphertext = plaintext;
    for (char& c : ciphertext) {
        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            c = ((c - base + shift) % 26) + base; // Apply the Caesar cipher shift
        }
    }
    return ciphertext;
}

// Function to perform row transposition encryption
```

```
string rowTranspositionEncrypt(const string& plaintext, const vector<int>& key) {
    int numRows = key.size();
    int numCols = (plaintext.length() + numRows - 1) / numRows;
    vector<vector<char>> grid(numRows, vector<char>(numCols, ' '));

    int index = 0;
    for (int col = 0; col < numCols; ++col) {
        for (int row = 0; row < numRows; ++row) {
            if (index < plaintext.length()) {
                grid[row][col] = plaintext[index++];
            }
        }
    }

    string ciphertext;
    for (int row : key) {
        for (int col = 0; col < numCols; ++col) {
            if (grid[row][col] != ' ') {
                ciphertext += grid[row][col];
            }
        }
    }

    return ciphertext;
}

// Function to perform product cipher encryption
string productCipherEncrypt(const string& plaintext, int caesarShift, const vector<int>&
transpositionKey) {
    string caesarEncrypted = caesarEncrypt(plaintext, caesarShift);
    string ciphertext = rowTranspositionEncrypt(caesarEncrypted, transpositionKey);
    return ciphertext;
}

int main() {
    string plaintext = "HELLO WORLD";
    int caesarShift = 3; // Caesar cipher shift
    vector<int> transpositionKey = {1, 0, 3, 2}; // Transposition key

    // Encrypt using product cipher
    string ciphertext = productCipherEncrypt(plaintext, caesarShift, transpositionKey);
    cout << "Encrypted text: " << ciphertext << endl;

    return 0;
}
```

Output-

Practical File

Output-

Q11- Implement a stream cipher technique.

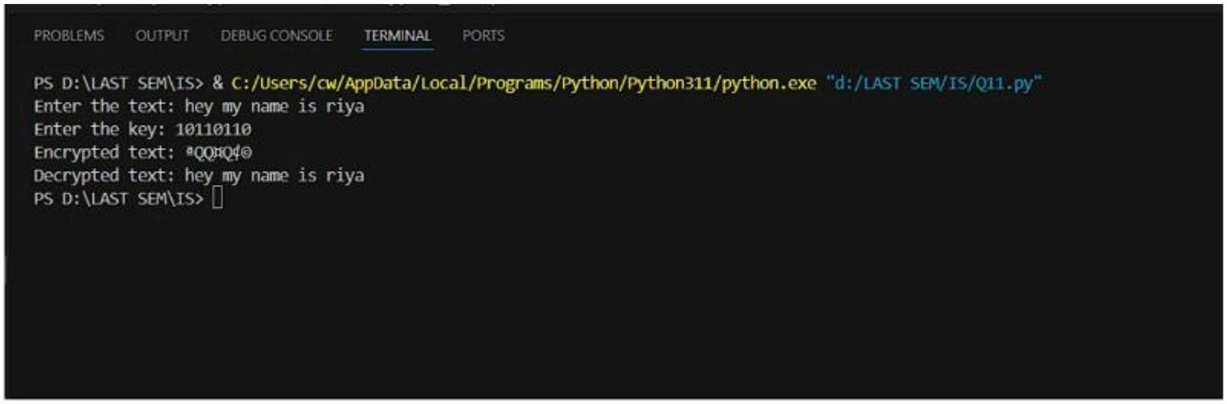
Code-

```
def stream_cipher(text, key, mode):
key_length = len(key)
text_length = len(text)
result = ""
for i in range(text_length):
key_index = i % key_length
```

```
if mode == 'encrypt':
result += chr((ord(text[i]) + ord(key[key_index])) % 256)
elif mode == 'decrypt':
result += chr((ord(text[i]) - ord(key[key_index])) % 256)
return result
# Take input from the user
text = input("Enter the text: ")
key = input("Enter the key: ")
# Encrypt the text
encrypted_text = stream_cipher(text, key, 'encrypt')
print("Encrypted text:", encrypted_text)
# Decrypt the encrypted text
decrypted_text = stream_cipher(encrypted_text, key, 'decrypt')
print("Decrypted text:", decrypted_text)
```

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

// Function to perform XOR encryption
string xorEncrypt(const string& plaintext, const string& key) {
   string ciphertext = plaintext;
   for (size_t i = 0; i < plaintext.size(); ++i) {
      ciphertext[i] = plaintext[i] ^ key[i % key.size()]; // XOR with corresponding key byte
   }
   return ciphertext;
}

// Function to perform XOR decryption
string xorDecrypt(const string& ciphertext, const string& key) {
   return xorEncrypt(ciphertext, key); // XOR encryption and decryption are the same operation
}
```

```
int main() {
    string plaintext = "Hello, world!";
    string key = "secret"; // Example key

    // Encrypt
    string encryptedText = xorEncrypt(plaintext, key);
    cout << "Encrypted text: " << encryptedText << endl;

    // Decrypt
    string decryptedText = xorDecrypt(encryptedText, key);
    cout << "Decrypted text: " << decryptedText << endl;

    return 0;
}
```



```
PS D:\LAST SEM\IS> & C:/Users/cw/AppData/Local/Programs/Python/Python311/python.exe "d:/LAST SEM/IS/Q11.py"
Enter the text: hey my name is riya
Enter the key: 10110110
Encrypted text: ªQQ¤Q¢©
Decrypted text: hey my name is riya
PS D:\LAST SEM\IS> 
```