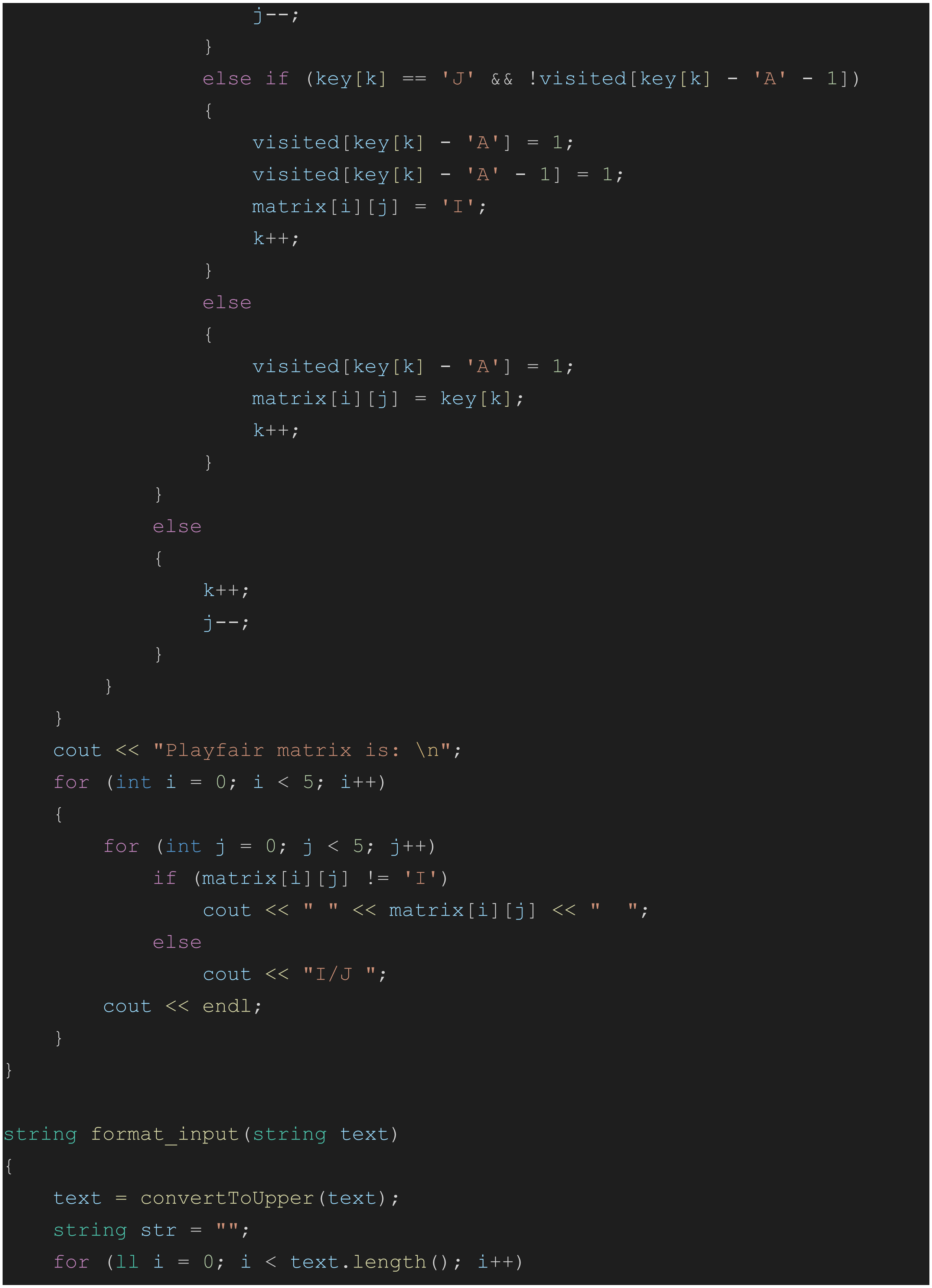
Computer Science and Engineering Department, SVNIT, Surat
ASSIGNMENT- 2

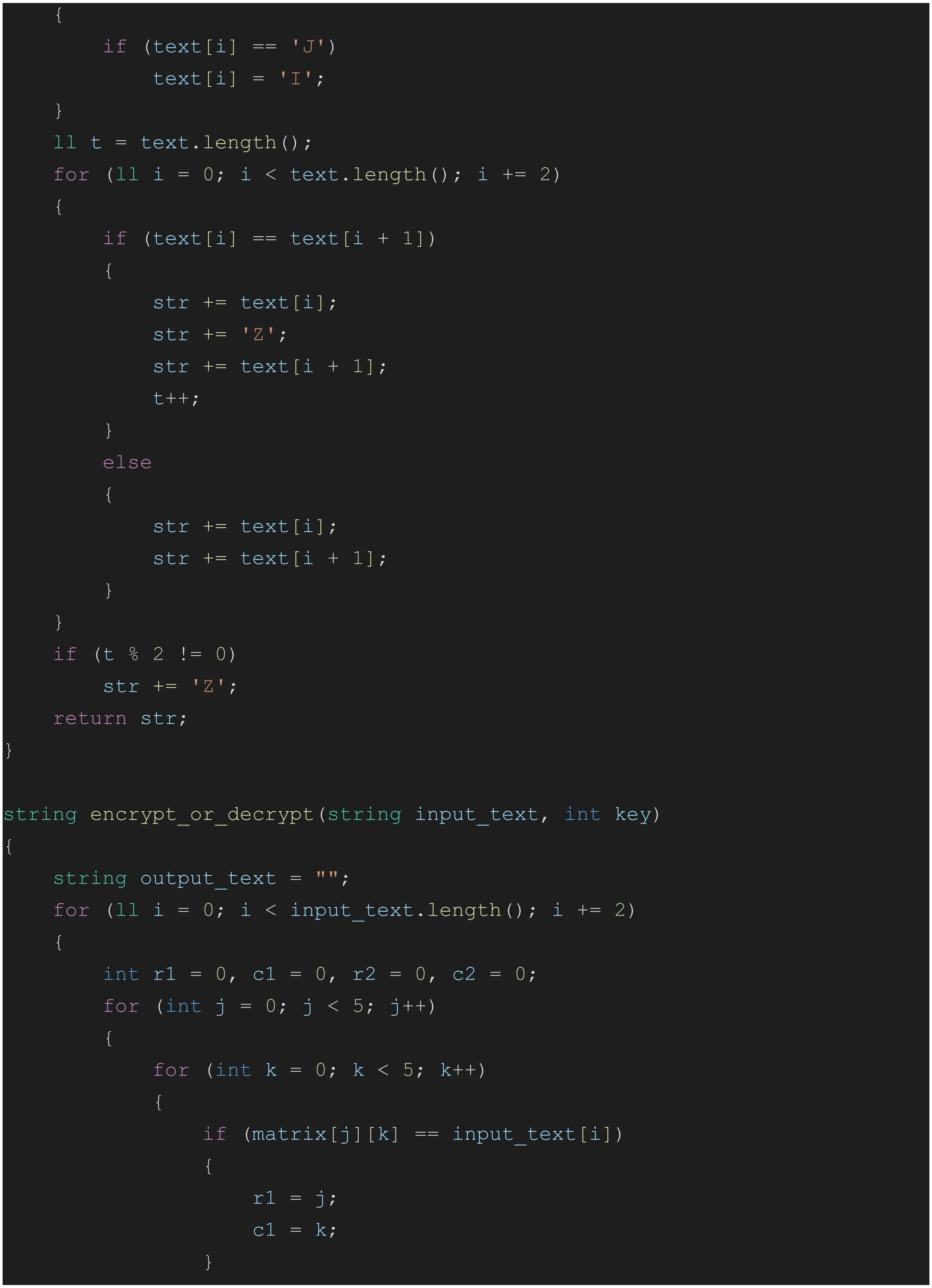
U20CS005
BANSI MARAKANA

Implement a menu driven program for 5X5 Playfair Cipher with following functions.
1. Takes text phrases to generate a key matrix.
2. Encrypt given plain text.
3. Decrypt given ciphertext.

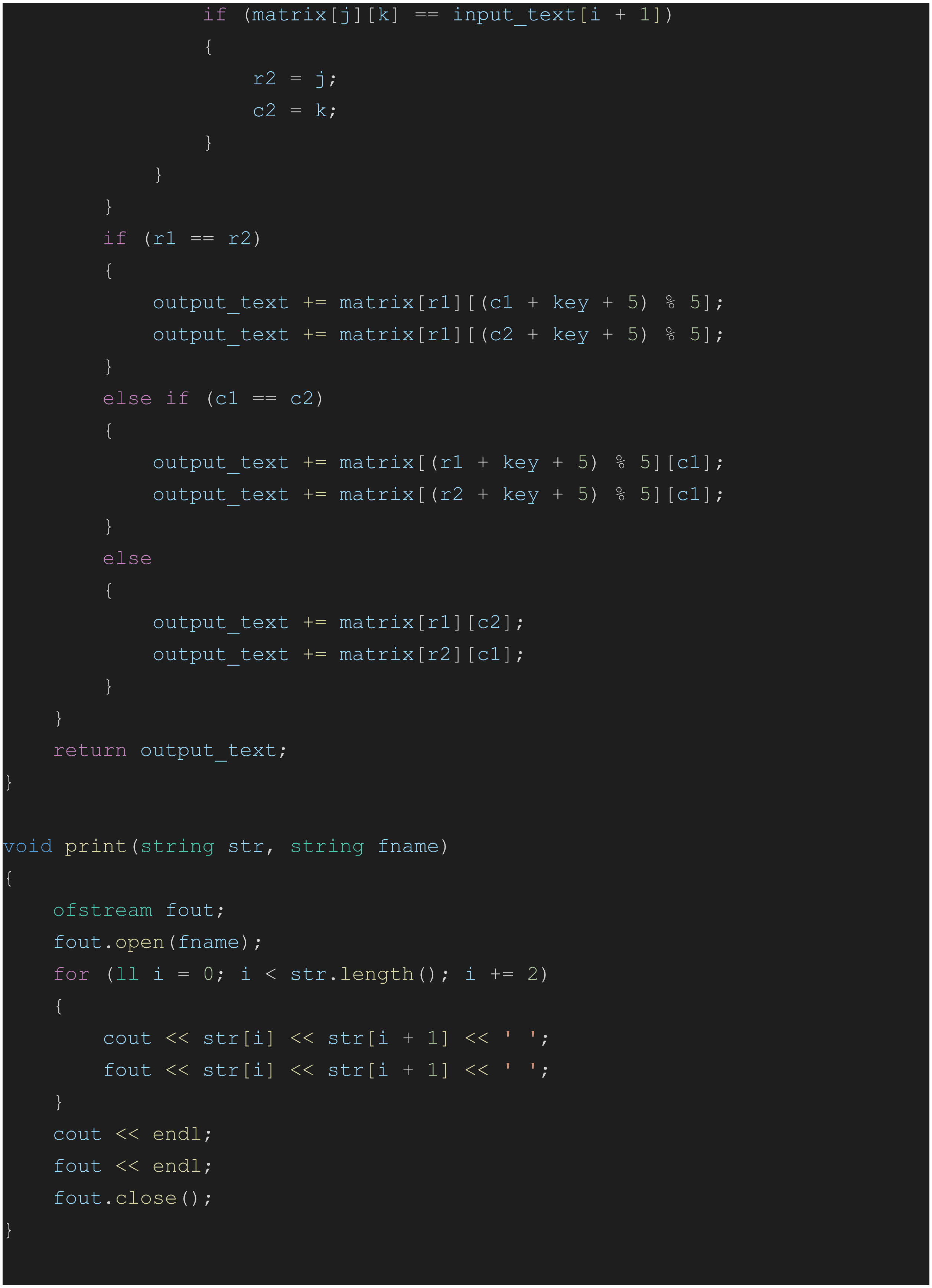
```
#include <bits/stdc++.h>
#include <iostream>
#include <fstream>
using namespace std;
typedef long long ll;
char matrix[5][5];

string convertToUpper(string text)
{
    text.erase(remove(text.begin(), text.end(), ' '), text.end());
    ll length = text.size();
    for (int i = 0; i < length; i++)
        text[i] = toupper(text[i]);
    return text;
}

void play_matrix(string key)
{
    key = convertToUpper(key);
    key += "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    int visited[26] = {0}, k = 0;
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            if (!visited[key[k] - 'A'] && matrix[i][j] == 0)
            {
                if (key[k] == 'J' && visited[key[k] - 'A' - 1])
                {
                    visited[key[k] - 'A'] = 1;
                    k++;
```



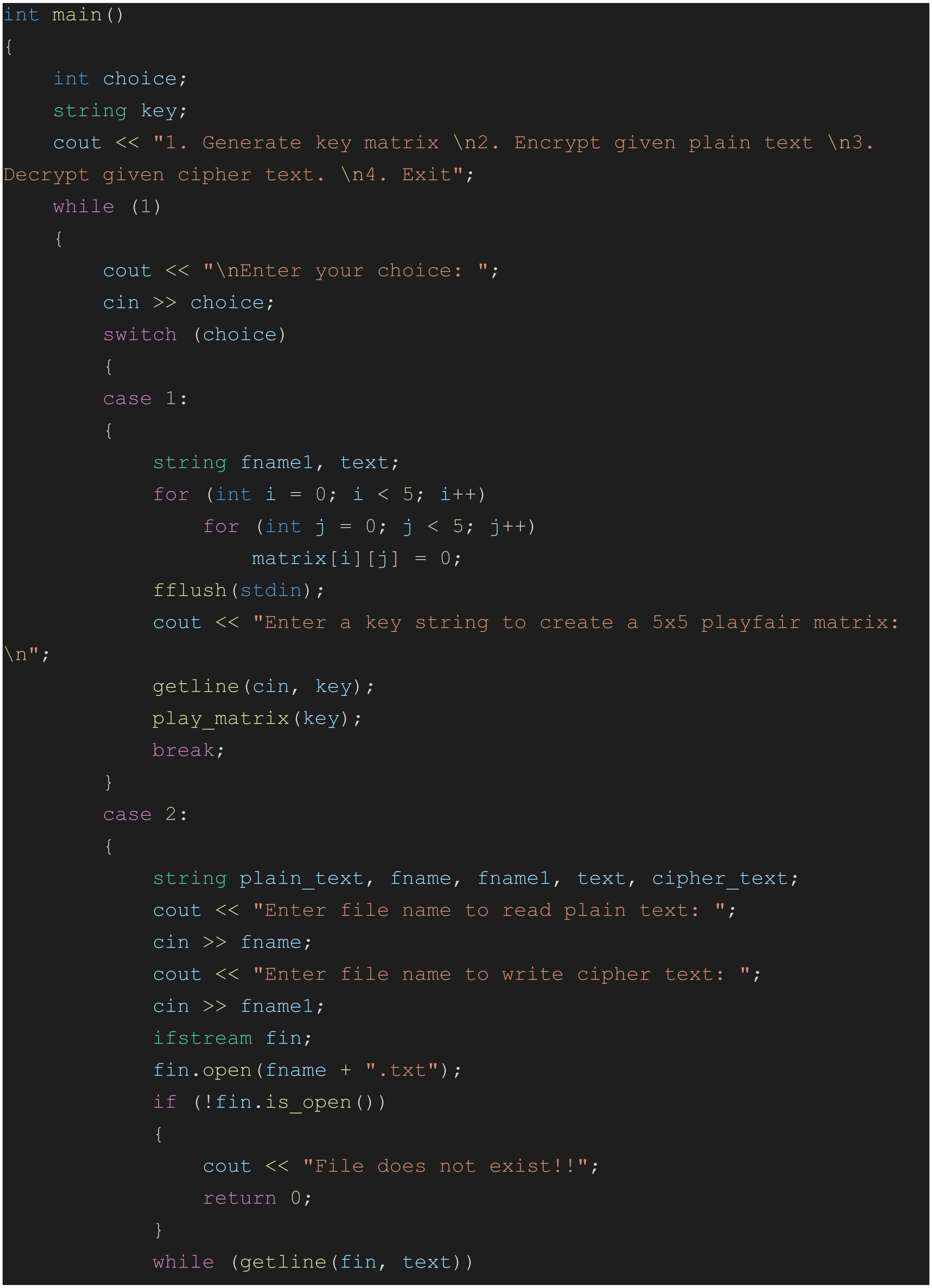
```
                    j--;
                }
                else if (key[k] == 'J' && !visited[key[k] - 'A' - 1])
                {
                    visited[key[k] - 'A'] = 1;
                    visited[key[k] - 'A' - 1] = 1;
                    matrix[i][j] = 'I';
                    k++;
                }
                else
                {
                    visited[key[k] - 'A'] = 1;
                    matrix[i][j] = key[k];
                    k++;
                }
            }
            else
            {
                k++;
                j--;
            }
        }
    }
    cout << "Playfair matrix is: \n";
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
            if (matrix[i][j] != 'I')
                cout << " " << matrix[i][j] << "  ";
            else
                cout << "I/J ";
        cout << endl;
    }
}

string format_input(string text)
{
    text = convertToUpper(text);
    string str = "";
    for (ll i = 0; i < text.length(); i++)
```



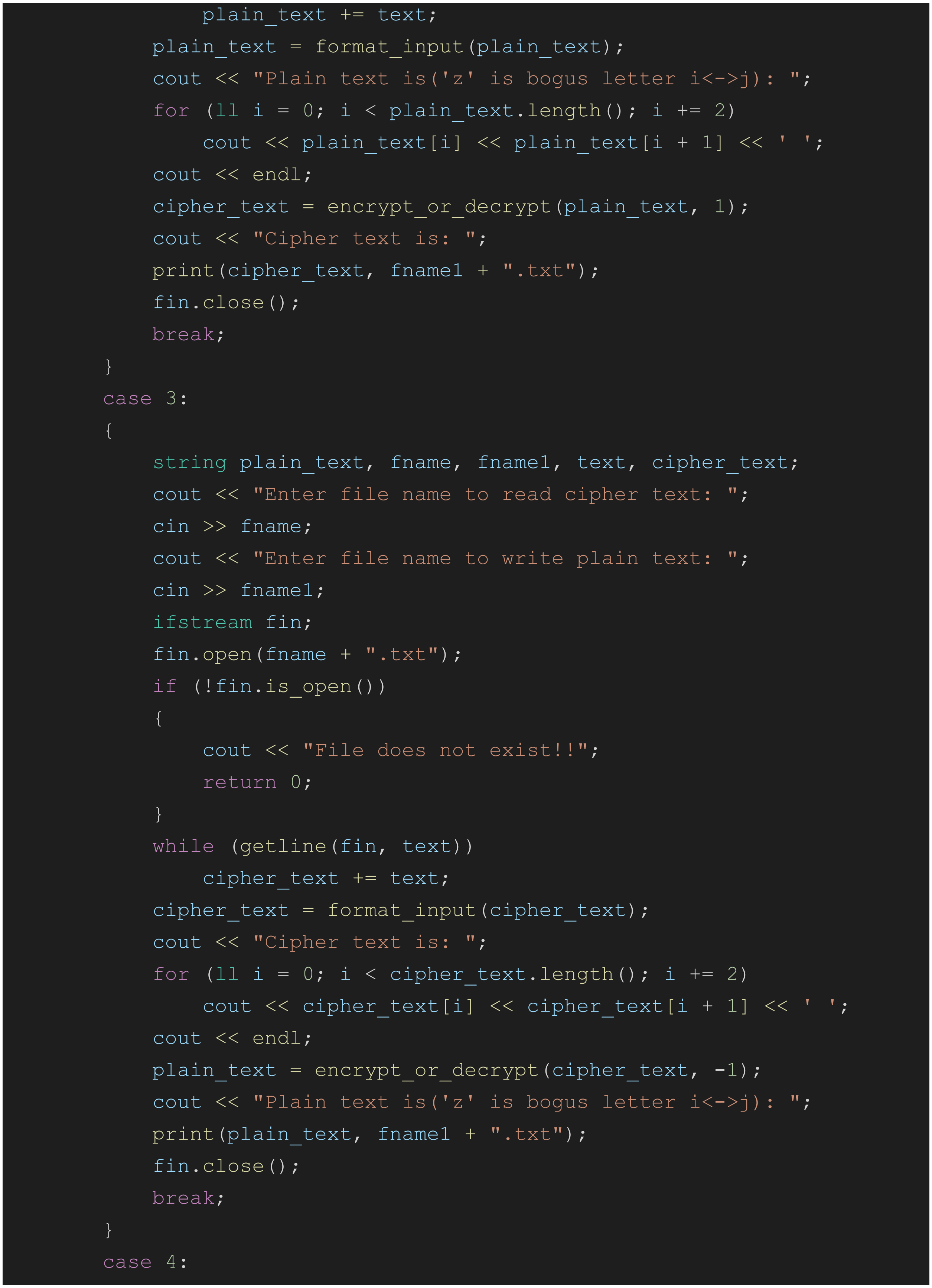
```
    {
        if (text[i] == 'J')
            text[i] = 'I';
    }
    ll t = text.length();
    for (ll i = 0; i < text.length(); i += 2)
    {
        if (text[i] == text[i + 1])
        {
            str += text[i];
            str += 'Z';
            str += text[i + 1];
            t++;
        }
        else
        {
            str += text[i];
            str += text[i + 1];
        }
    }
    if (t % 2 != 0)
        str += 'Z';
    return str;
}

string encrypt_or_decrypt(string input_text, int key)
{
    string output_text = "";
    for (ll i = 0; i < input_text.length(); i += 2)
    {
        int r1 = 0, c1 = 0, r2 = 0, c2 = 0;
        for (int j = 0; j < 5; j++)
        {
            for (int k = 0; k < 5; k++)
            {
                if (matrix[j][k] == input_text[i])
                {
                    r1 = j;
                    c1 = k;
                }
```



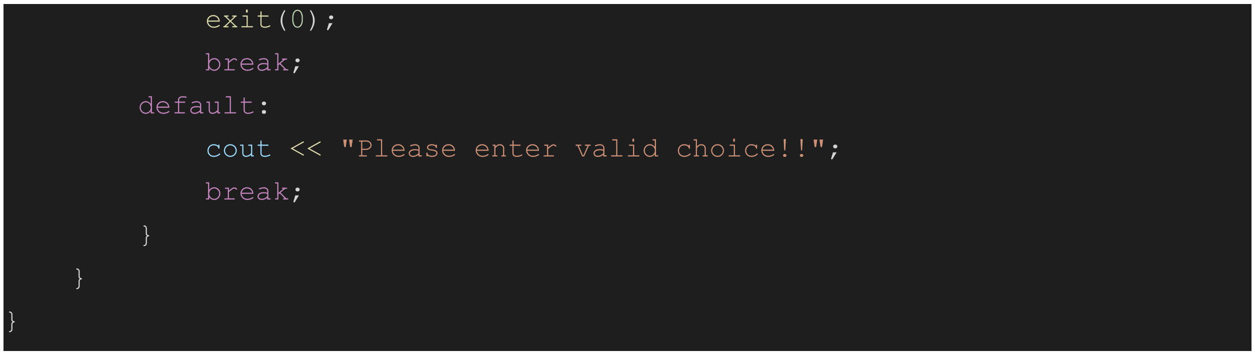
```
                if (matrix[j][k] == input_text[i + 1])
                {
                    r2 = j;
                    c2 = k;
                }
            }
        }
        if (r1 == r2)
        {
            output_text += matrix[r1][(c1 + key + 5) % 5];
            output_text += matrix[r1][(c2 + key + 5) % 5];
        }
        else if (c1 == c2)
        {
            output_text += matrix[(r1 + key + 5) % 5][c1];
            output_text += matrix[(r2 + key + 5) % 5][c1];
        }
        else
        {
            output_text += matrix[r1][c2];
            output_text += matrix[r2][c1];
        }
    }
    return output_text;
}

void print(string str, string fname)
{
    ofstream fout;
    fout.open(fname);
    for (ll i = 0; i < str.length(); i += 2)
    {
        cout << str[i] << str[i + 1] << ' ';
        fout << str[i] << str[i + 1] << ' ';
    }
    cout << endl;
    fout << endl;
    fout.close();
}
```



```
int main()
{
    int choice;
    string key;
    cout << "1. Generate key matrix \n2. Encrypt given plain text \n3.
Decrypt given cipher text. \n4. Exit";
    while (1)
    {
        cout << "\nEnter your choice: ";
        cin >> choice;
        switch (choice)
        {
        case 1:
        {
            string fname1, text;
            for (int i = 0; i < 5; i++)
                for (int j = 0; j < 5; j++)
                    matrix[i][j] = 0;
            fflush(stdin);
            cout << "Enter a key string to create a 5x5 playfair matrix:
\n";
            getline(cin, key);
            play_matrix(key);
            break;
        }
        case 2:
        {
            string plain_text, fname, fname1, text, cipher_text;
            cout << "Enter file name to read plain text: ";
            cin >> fname;
            cout << "Enter file name to write cipher text: ";
            cin >> fname1;
            ifstream fin;
            fin.open(fname + ".txt");
            if (!fin.is_open())
            {
                cout << "File does not exist!!";
                return 0;
            }
            while (getline(fin, text))
```



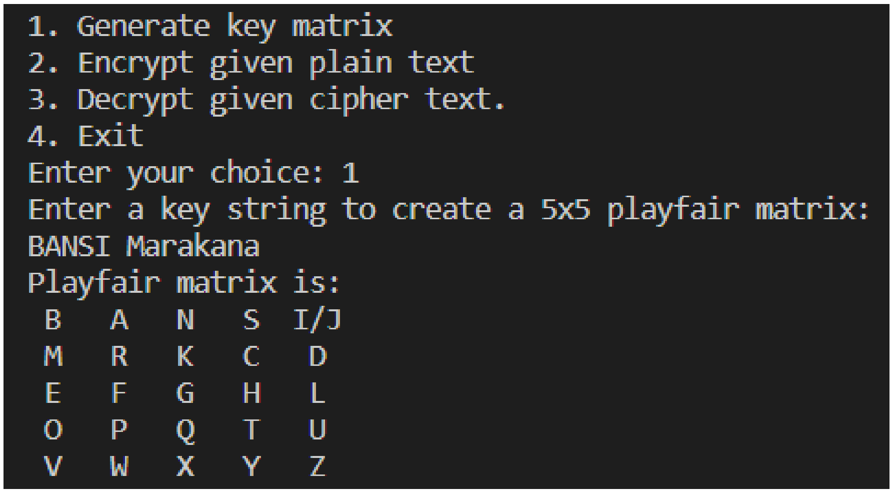
```
                plain_text += text;
            plain_text = format_input(plain_text);
            cout << "Plain text is('z' is bogus letter i<->j): ";
            for (ll i = 0; i < plain_text.length(); i += 2)
                cout << plain_text[i] << plain_text[i + 1] << ' ';
            cout << endl;
            cipher_text = encrypt_or_decrypt(plain_text, 1);
            cout << "Cipher text is: ";
            print(cipher_text, fname1 + ".txt");
            fin.close();
            break;
        }
        case 3:
        {
            string plain_text, fname, fname1, text, cipher_text;
            cout << "Enter file name to read cipher text: ";
            cin >> fname;
            cout << "Enter file name to write plain text: ";
            cin >> fname1;
            ifstream fin;
            fin.open(fname + ".txt");
            if (!fin.is_open())
            {
                cout << "File does not exist!!";
                return 0;
            }
            while (getline(fin, text))
                cipher_text += text;
            cipher_text = format_input(cipher_text);
            cout << "Cipher text is: ";
            for (ll i = 0; i < cipher_text.length(); i += 2)
                cout << cipher_text[i] << cipher_text[i + 1] << ' ';
            cout << endl;
            plain_text = encrypt_or_decrypt(cipher_text, -1);
            cout << "Plain text is('z' is bogus letter i<->j): ";
            print(plain_text, fname1 + ".txt");
            fin.close();
            break;
        }
        case 4:
```



```
            exit(0);
            break;
        default:
            cout << "Please enter valid choice!!";
            break;
        }
    }
}
```

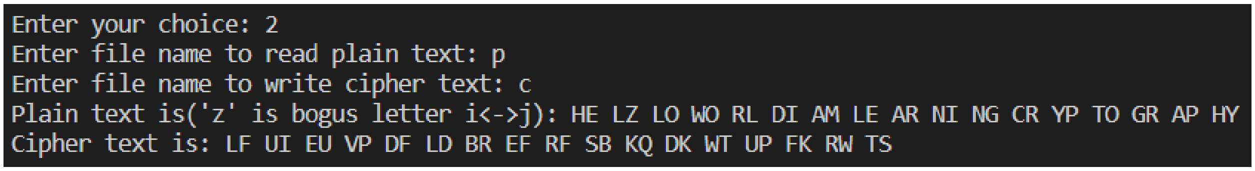
OUTPUT:

a. Key Matrix:



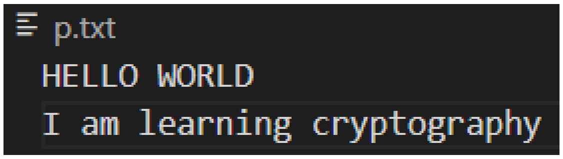
```
1. Generate key matrix
2. Encrypt given plain text
3. Decrypt given cipher text.
4. Exit
Enter your choice: 1
Enter a key string to create a 5x5 playfair matrix:
BANSI Marakana
Playfair matrix is:
 B   A   N   S  I/J
 M   R   K   C   D
 E   F   G   H   L
 O   P   Q   T   U
 V   W   X   Y   Z
```

b. Encryption



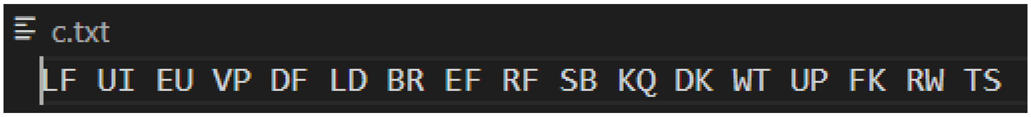
```
Enter your choice: 2
Enter file name to read plain text: p
Enter file name to write cipher text: c
Plain text is('z' is bogus letter i<->j): HE LZ LO WO RL DI AM LE AR NI NG CR YP TO GR AP HY
Cipher text is: LF UI EU VP DF LD BR EF RF SB KQ DK WT UP FK RW TS
```

p.txt:



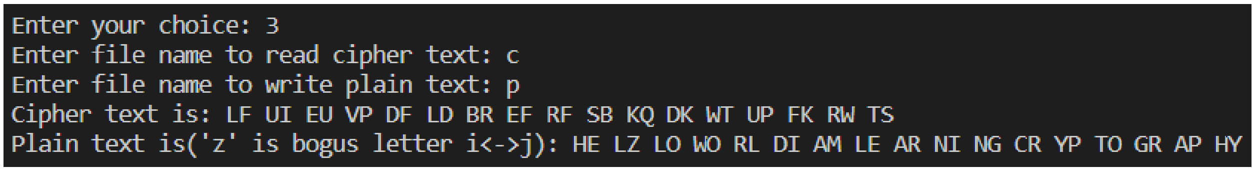
```
p.txt
HELLO WORLD
I am learning cryptography
```

c.txt:



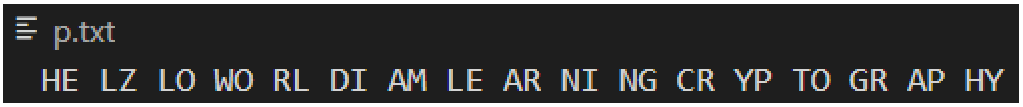
```
c.txt
LF UI EU VP DF LD BR EF RF SB KQ DK WT UP FK RW TS
```

c. Decryption:



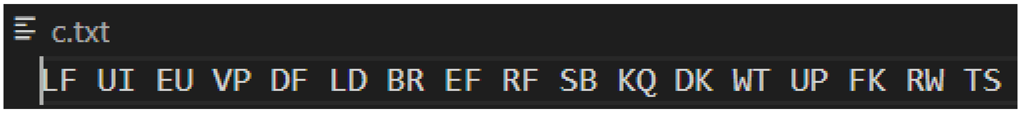
```
Enter your choice: 3
Enter file name to read cipher text: c
Enter file name to write plain text: p
Cipher text is: LF UI EU VP DF LD BR EF RF SB KQ DK WT UP FK RW TS
Plain text is('z' is bogus letter i<->j): HE LZ LO WO RL DI AM LE AR NI NG CR YP TO GR AP HY
```

p.txt:



```
HE LZ LO WO RL DI AM LE AR NI NG CR YP TO GR AP HY
```

c.txt:



```
LF UI EU VP DF LD BR EF RF SB KQ DK WT UP FK RW TS
```