Name: Rutikesh Sawant
Batch: B2
Subject: CNS Lab
PRN: 2019BTECS00034

Assignment 3

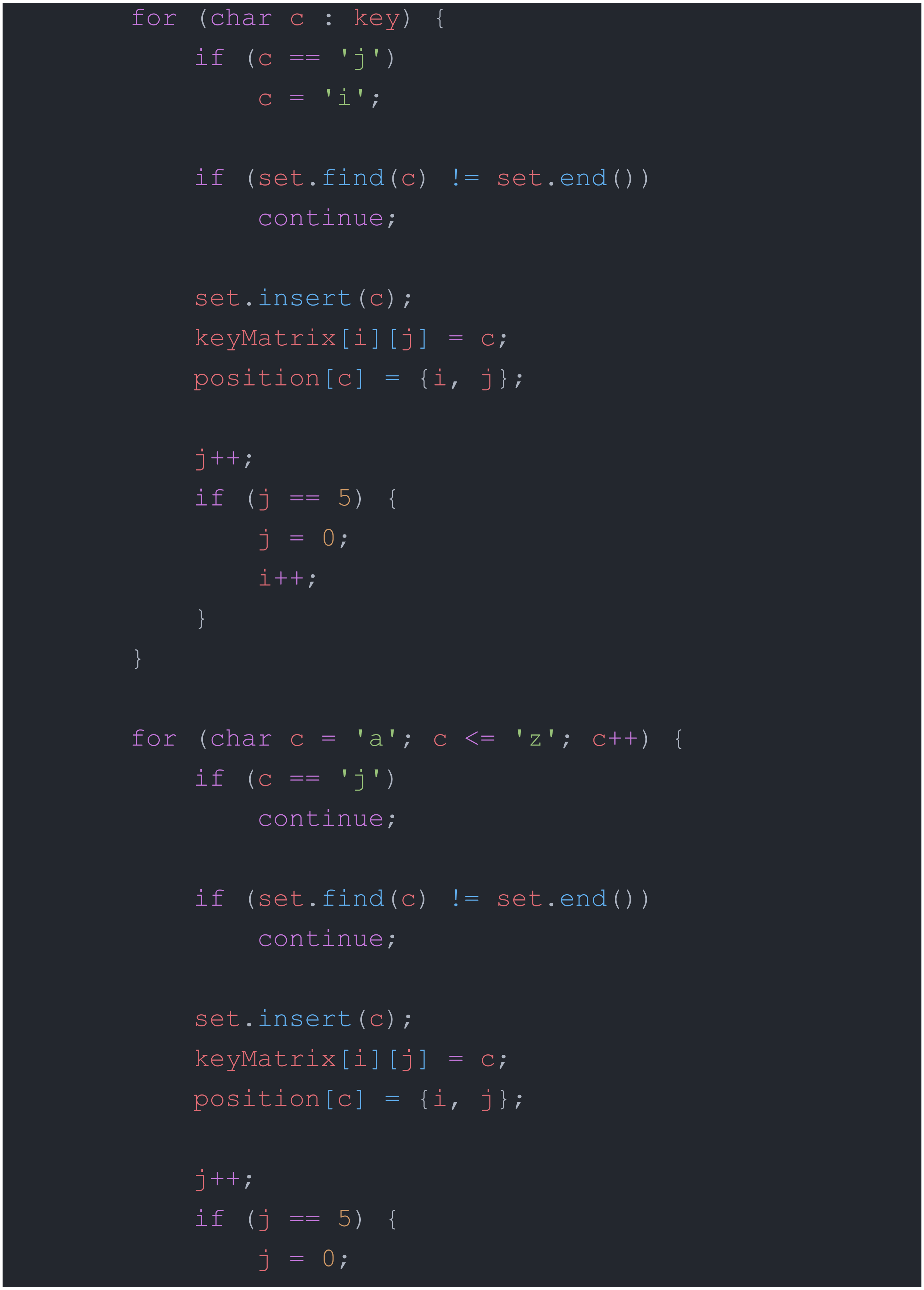
Aim: To encrypt plain text using PlayFair cipher and decrypt the cipher text to plain text.

Theory:

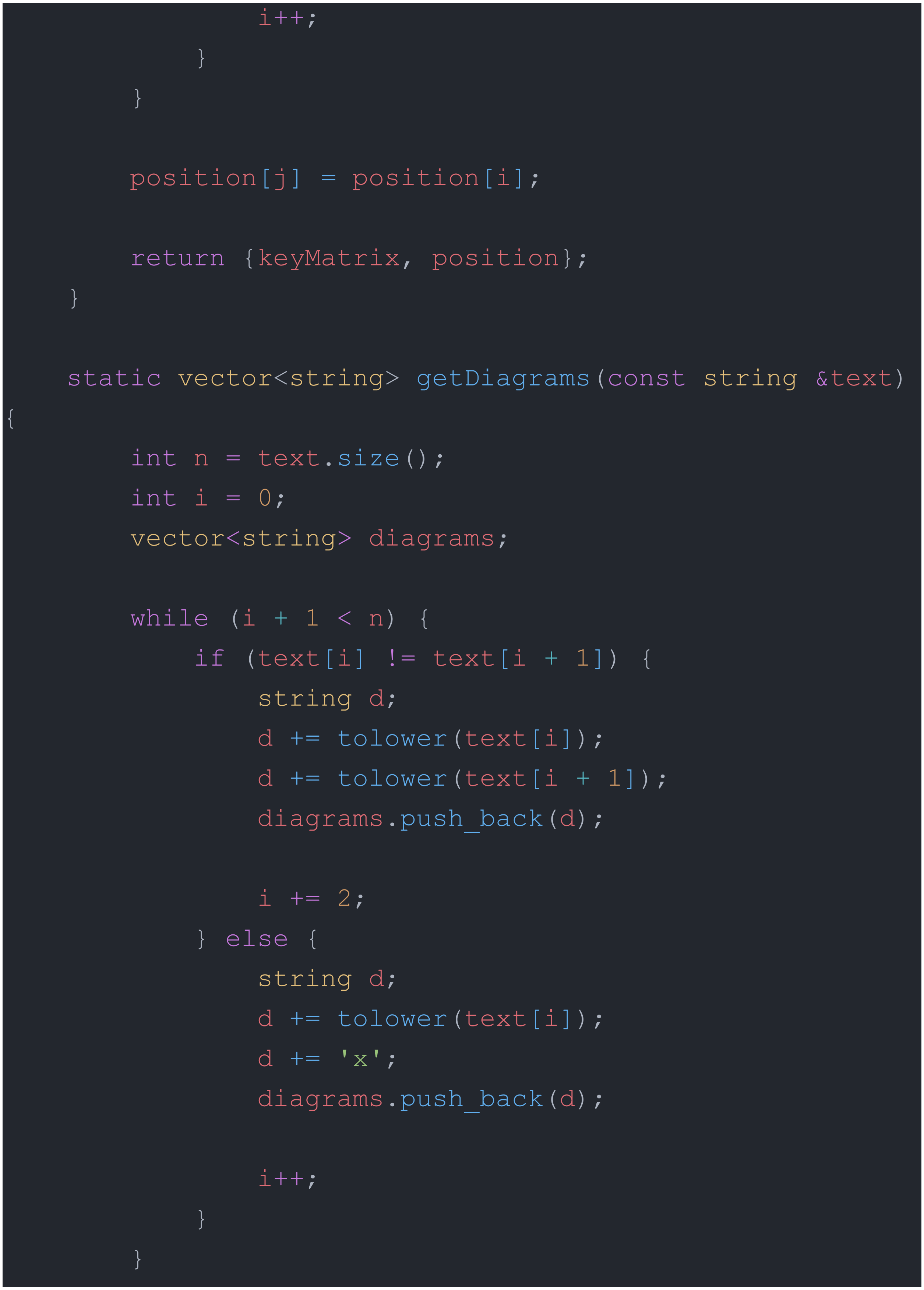
Playfair cipher is a manual symmetric encryption technique and was first diagram substitution cipher. In playfair cipher group of letters is encrypted instead of a single letter so it is little bit complex than caesar cipher. So it is hard to break playfair cipher algorithm as in simple caesar cipher one can easily predict k value and decrypt the text easily. So this playfair cipher algorithm is more secure than caesar cipher.

Code:

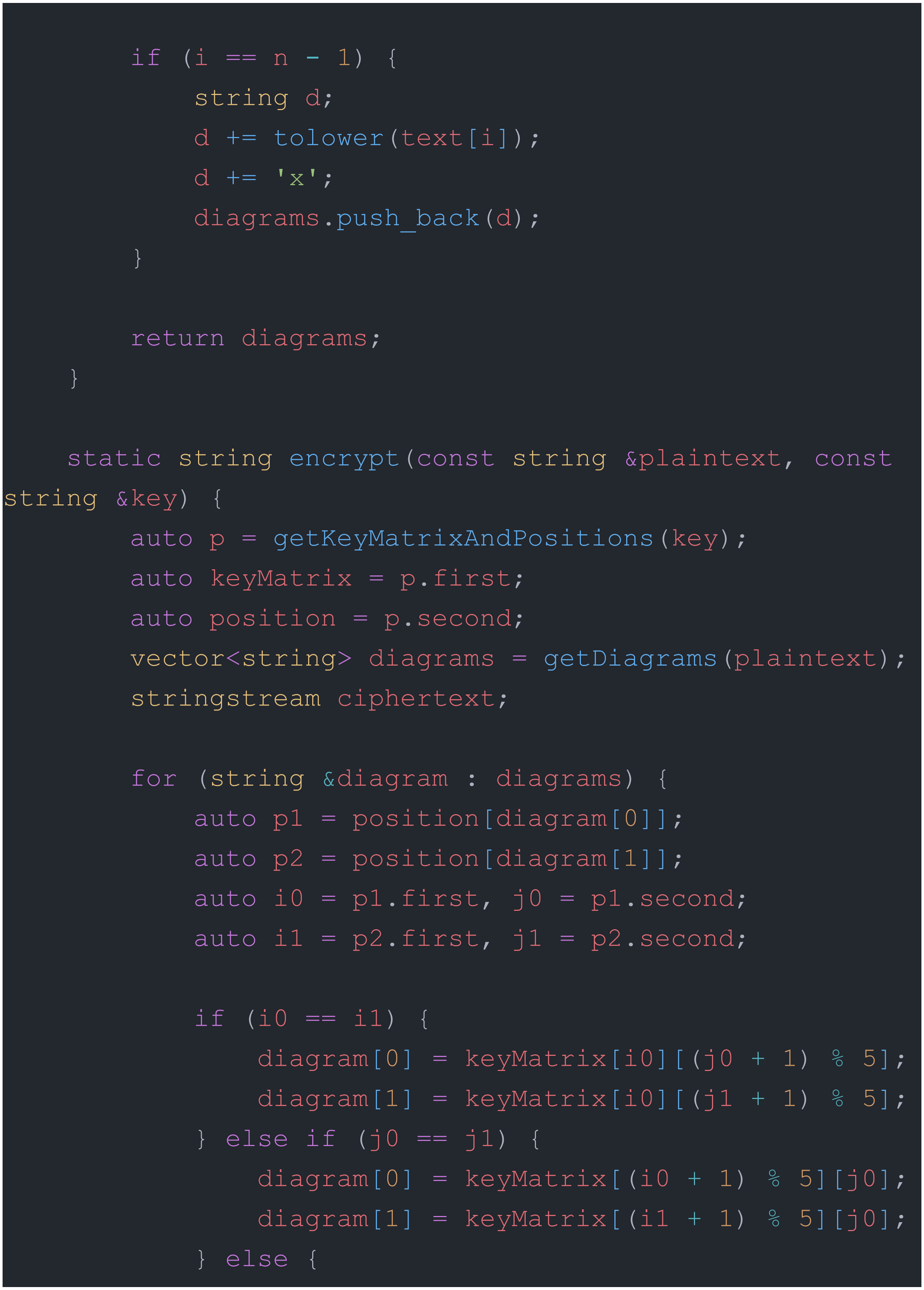
```
#include <bits/stdc++.h>
using namespace std;

class PlayfairCipher {
  public:
    static pair<vector<vector<char>>, unordered_map<char,
pair<int, int>>>
    getKeyMatrixAndPositions(const string &key) {
        vector<vector<char>> keyMatrix(5,
vector<char>(5));
        int i = 0, j = 0;
        unordered_set<char> set;
        unordered_map<char, pair<int, int>> position;

```



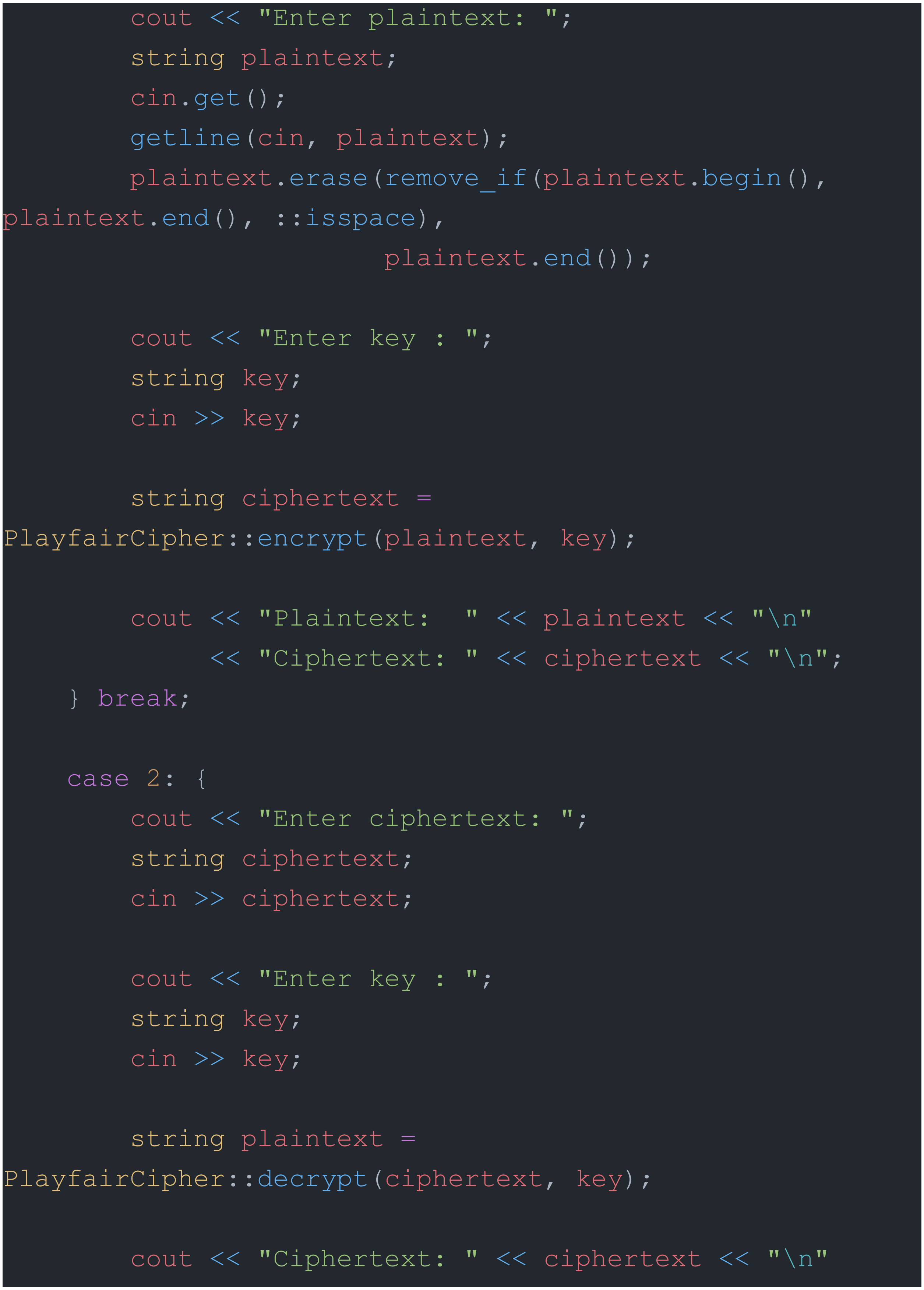
```
        for (char c : key) {
            if (c == 'j')
                c = 'i';

            if (set.find(c) != set.end())
                continue;

            set.insert(c);
            keyMatrix[i][j] = c;
            position[c] = {i, j};

            j++;
            if (j == 5) {
                j = 0;
                i++;
            }
        }

        for (char c = 'a'; c <= 'z'; c++) {
            if (c == 'j')
                continue;

            if (set.find(c) != set.end())
                continue;

            set.insert(c);
            keyMatrix[i][j] = c;
            position[c] = {i, j};

            j++;
            if (j == 5) {
                j = 0;
```



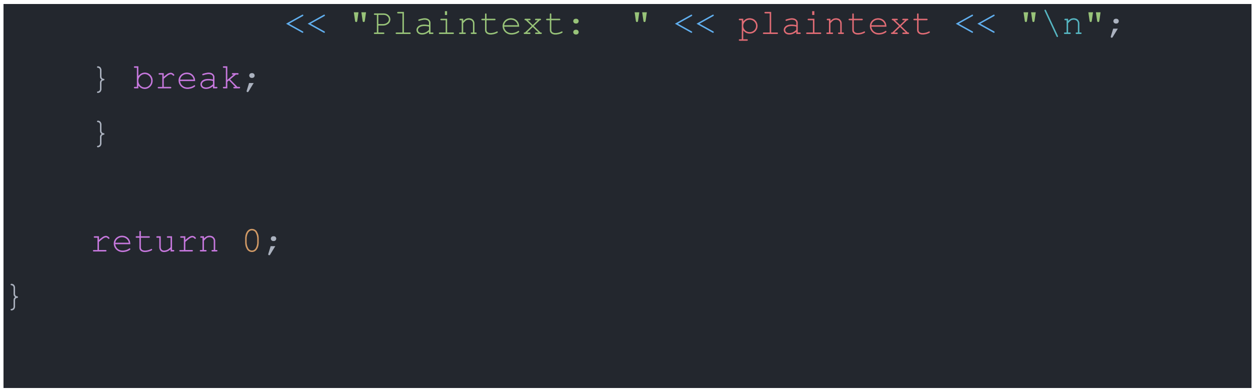
```
                i++;
            }
        }

        position[j] = position[i];

        return {keyMatrix, position};
    }

    static vector<string> getDiagrams(const string &text)
{
        int n = text.size();
        int i = 0;
        vector<string> diagrams;

        while (i + 1 < n) {
            if (text[i] != text[i + 1]) {
                string d;
                d += tolower(text[i]);
                d += tolower(text[i + 1]);
                diagrams.push_back(d);

                i += 2;
            } else {
                string d;
                d += tolower(text[i]);
                d += 'x';
                diagrams.push_back(d);

                i++;
            }
        }
```



```
        if (i == n - 1) {
            string d;
            d += tolower(text[i]);
            d += 'x';
            diagrams.push_back(d);
        }

        return diagrams;
    }

    static string encrypt(const string &plaintext, const
string &key) {
        auto p = getKeyMatrixAndPositions(key);
        auto keyMatrix = p.first;
        auto position = p.second;
        vector<string> diagrams = getDiagrams(plaintext);
        stringstream ciphertext;

        for (string &diagram : diagrams) {
            auto p1 = position[diagram[0]];
            auto p2 = position[diagram[1]];
            auto i0 = p1.first, j0 = p1.second;
            auto i1 = p2.first, j1 = p2.second;

            if (i0 == i1) {
                diagram[0] = keyMatrix[i0][(j0 + 1) % 5];
                diagram[1] = keyMatrix[i0][(j1 + 1) % 5];
            } else if (j0 == j1) {
                diagram[0] = keyMatrix[(i0 + 1) % 5][j0];
                diagram[1] = keyMatrix[(i1 + 1) % 5][j0];
            } else {
                diagram[0] = keyMatrix[i0][j1];
                diagram[1] = keyMatrix[i1][j0];
            }

            ciphertext << diagram;
        }

        string answer = ciphertext.str();

        transform(answer.begin(), answer.end(),
answer.begin(), ::toupper);
        return answer;
    }

    static string decrypt(const string &ciphertext, const
string &key) {
        auto p = getKeyMatrixAndPositions(key);
        auto keyMatrix = p.first;
        auto position = p.second;

        vector<string> diagrams =
getDiagrams(ciphertext);
        stringstream plaintext;

        for (string &diagram : diagrams) {
            auto p1 = position[diagram[0]];
            auto p2 = position[diagram[1]];
            auto i0 = p1.first, j0 = p1.second;
            auto i1 = p2.first, j1 = p2.second;

            if (i0 == i1) {
                diagram[0] = keyMatrix[i0][(j0 - 1 + 5) %
5];
                diagram[1] = keyMatrix[i0][(j1 - 1 + 5) %
5];
            } else if (j0 == j1) {
                diagram[0] = keyMatrix[(i0 - 1 + 5) %
5][j0];
                diagram[1] = keyMatrix[(i1 - 1 + 5) %
5][j0];
            } else {
                diagram[0] = keyMatrix[i0][j1];
                diagram[1] = keyMatrix[i1][j0];
            }

            plaintext << diagram;
        }

        return plaintext.str();
    }
};

int main() {
    cout << "PlayFair Cipher:\n"
         << "Enter your choice:\n"
         << "1. Encrypt\n"
         << "2. Decrypt\n";

    int choice;
    cin >> choice;

    switch (choice) {
    case 1: {
```

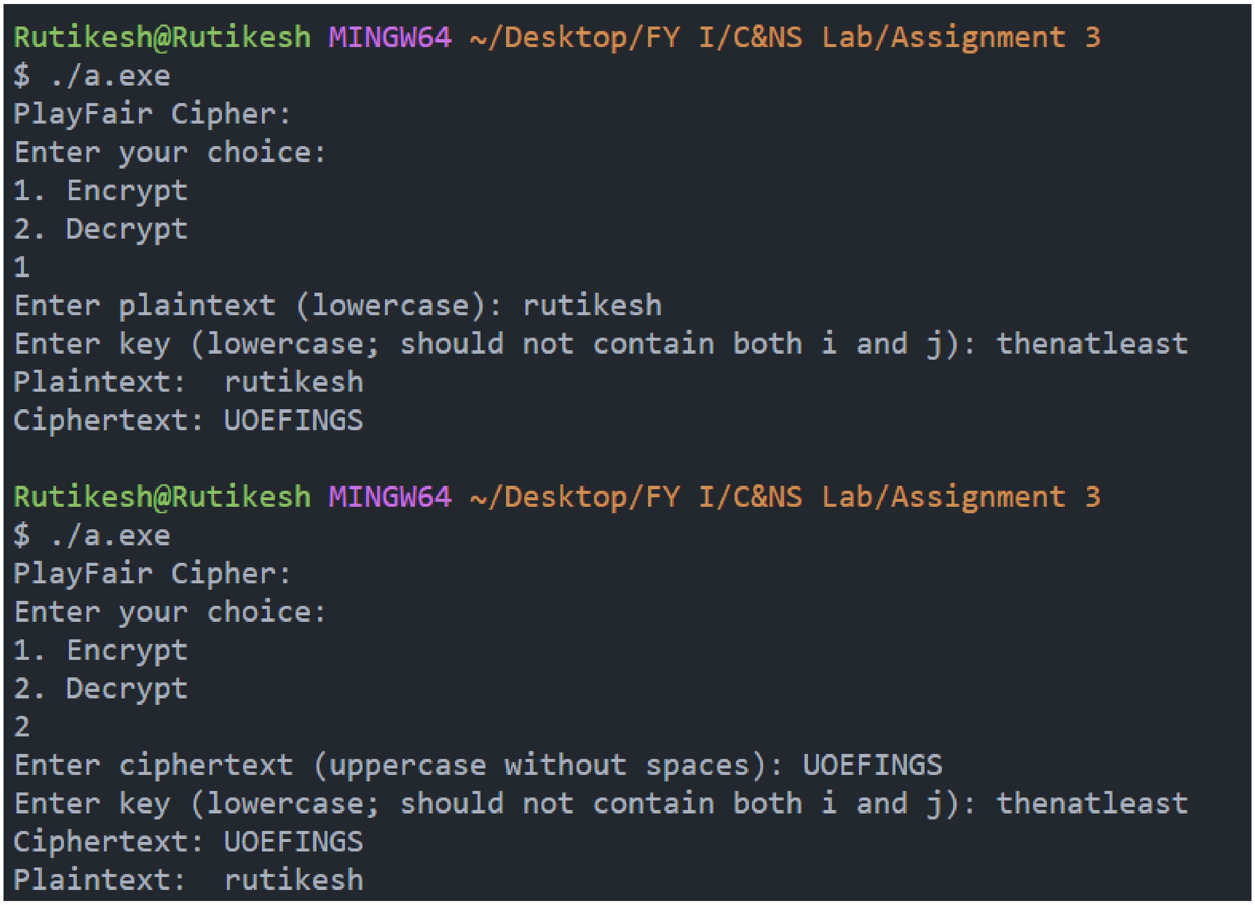


```
        cout << "Enter plaintext: ";
        string plaintext;
        cin.get();
        getline(cin, plaintext);
        plaintext.erase(remove_if(plaintext.begin(),
plaintext.end(), ::isspace),
                        plaintext.end());

        cout << "Enter key : ";
        string key;
        cin >> key;

        string ciphertext =
PlayfairCipher::encrypt(plaintext, key);

        cout << "Plaintext:  " << plaintext << "\n"
             << "Ciphertext: " << ciphertext << "\n";
    } break;

    case 2: {
        cout << "Enter ciphertext: ";
        string ciphertext;
        cin >> ciphertext;

        cout << "Enter key : ";
        string key;
        cin >> key;

        string plaintext =
PlayfairCipher::decrypt(ciphertext, key);

        cout << "Ciphertext: " << ciphertext << "\n"
```



```
             << "Plaintext:  " << plaintext << "\n";
    } break;
    }

    return 0;
}
```

Output:



```
Rutikesh@Rutikesh MINGW64 ~/Desktop/FY I/C&NS Lab/Assignment 3
$ ./a.exe
PlayFair Cipher:
Enter your choice:
1. Encrypt
2. Decrypt
1
Enter plaintext (lowercase): rutikesh
Enter key (lowercase; should not contain both i and j): thenatleast
Plaintext:  rutikesh
Ciphertext: UOEFINGS

Rutikesh@Rutikesh MINGW64 ~/Desktop/FY I/C&NS Lab/Assignment 3
$ ./a.exe
PlayFair Cipher:
Enter your choice:
1. Encrypt
2. Decrypt
2
Enter ciphertext (uppercase without spaces): UOEFINGS
Enter key (lowercase; should not contain both i and j): thenatleast
Ciphertext: UOEFINGS
Plaintext:  rutikesh
```