**Final Year B.Tech. (CSE) – II [ 2021-22 ]**

**Cryptograpy and Network Security Lab**

**PRN: 2019BTECS00015**

**Full name: Shraddha Sanjay Kharat**

**Batch: B1**

**Assignment no - 3**

**Date: 23-08-2022**

**Que )** **Playfair Cipher implementation**

**Aim:** Encrypting the cipher text using Playfair Cypher

**Theory:**

The Playfair Cipher Encryption Algorithm consists of 2 steps:

1) Generate the Key Square(5x5) :

The Key Square is a 5x% grid of alphabets that act as the key for encrypting the plain text. Each of the 25 alphabets must be unique and one letter of alphabet (Usually) J is omitted from the table. If the plaintext contains J, then it is replaced by I. The initial alphabets in the key square are the unique alphabets of the key in the order in which they appear followed by the remaining letters of the alphabet in order.

2) Algorithm to encrypt the plain text:

The plaintext is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter.

1. If both the letters are in same row : the letter to the right of each (going back to the leftmost position)
2. If both the letters are in same column: take the letter below of each one (going back to the top)
3. If neither of the above is true: form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle. Take the Row first.

**Code Snapshots:**

#include <bits/stdc++.h>

using namespace std;

int RemoveSpaces(char arr[], int len)

{

    int i, count = 0;

    for (i = 0; i < len; i++)

        if (arr[i] != ' ')

            arr[count++] = arr[i];

    arr[count] = '\0';

    return count;

}

void GenerateKeyTable(char keyarr[5], int keylen, char keyTable[5][5])

{

    int i, j, k, flag = 0;

    int map[26] = {0};

    for (i = 0; i < keylen; i++)

    {

        if (keyarr[i] != 'J')

            map[keyarr[i] - 65] = 2;

    }

    map['J' - 65] = 1;

    i = 0;

    j = 0;

    for (k = 0; k < keylen; k++)

    {

        if (map[keyarr[k] - 65] == 2)

        {

            map[keyarr[k] - 65] -= 1;

            keyTable[i][j] = keyarr[k];

            j++;

            if (j == 5)

            {

                i++;

                j = 0;

            }

        }

    }

    for (k = 0; k < 26; k++)

    {

        if (map[k] == 0)

        {

            keyTable[i][j] = (char)(k + 65);

            j++;

            if (j == 5)

            {

                i++;

                j = 0;

            }

        }

    }

}

void search(char keyTable[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 (keyTable[i][j] == a)

            {

                arr[0] = i;

                arr[1] = j;

            }

            else if (keyTable[i][j] == b)

            {

                arr[2] = i;

                arr[3] = j;

            }

        }

    }

}

int mod5(int a)

{

    if (a < 0)

        a += 5;

    return (a % 5);

}

int Prepare(char arr[], int len)

{

    string res;

    int i = 0;

    while (i < len - 1)

    {

        res.push\_back(arr[i]);

        if (arr[i] == arr[i + 1])

        {

            res.push\_back('X');

            i++;

        }

        else

        {

            res.push\_back(arr[i + 1]);

            i += 2;

        }

    }

    len = res.size();

    strcpy(arr, res.c\_str());

    if (len % 2 != 0)

    {

        arr[len++] = 'X';

        arr[len] = '\0';

    }

    return len;

}

void Encrypt(char textarr[], char keyTable[5][5], int textlen)

{

    int i, arr[4];

    for (i = 0; i < textlen; i += 2)

    {

        search(keyTable, textarr[i], textarr[i + 1], arr);

        if (arr[0] == arr[2])

        {

            textarr[i] = keyTable[arr[0]][mod5(arr[1] + 1)];

            textarr[i + 1] = keyTable[arr[0]][mod5(arr[3] + 1)];

        }

        else if (arr[1] == arr[3])

        {

            textarr[i] = keyTable[mod5(arr[0] + 1)][arr[1]];

            textarr[i + 1] = keyTable[mod5(arr[2] + 1)][arr[1]];

        }

        else

        {

            textarr[i] = keyTable[arr[0]][arr[3]];

            textarr[i + 1] = keyTable[arr[2]][arr[1]];

        }

    }

}

void Decrypt(char textarr[], char keyTable[5][5], int textlen)

{

    string res;

    int i, arr[4];

    for (i = 0; i < textlen; i += 2)

    {

        search(keyTable, textarr[i], textarr[i + 1], arr);

        if (arr[0] == arr[2])

        {

            textarr[i] = keyTable[arr[0]][mod5(arr[1] - 1)];

            textarr[i + 1] = keyTable[arr[0]][mod5(arr[3] - 1)];

        }

        else if (arr[1] == arr[3])

        {

            textarr[i] = keyTable[mod5(arr[0] - 1)][arr[1]];

            textarr[i + 1] = keyTable[mod5(arr[2] - 1)][arr[1]];

        }

        else

        {

            textarr[i] = keyTable[arr[0]][arr[3]];

            textarr[i + 1] = keyTable[arr[2]][arr[1]];

        }

    }

    for (i = 0; i < textlen; i++)

    {

        if(textarr[i]!='X') res.push\_back(textarr[i]);

    }

    strcpy(textarr, res.c\_str());

}

string PlayfairEncrypt(char textarr[], char keyarr[])

{

    char keyTable[5][5];

    string result;

    int keylen = RemoveSpaces(keyarr, strlen(keyarr));

    int textlen = RemoveSpaces(textarr, strlen(textarr));

    textlen = Prepare(textarr, textlen);

    GenerateKeyTable(keyarr, keylen, keyTable);

    Encrypt(textarr, keyTable, textlen);

    result = textarr;

    return result;

}

string PlayfairDecrypt(char textarr[], char keyarr[])

{

    char keyTable[5][5];

    string result;

    int keylen = strlen(keyarr);

    int textlen = strlen(textarr);

    GenerateKeyTable(keyarr, keylen, keyTable);

    Decrypt(textarr, keyTable, textlen);

    result = textarr;

    return result;

}

int main()

{

    int option;

    string key, text, cipherText;

    char keyarr[30], textarr[30], cipher[30];

    cout << "Enter option:\n1)Console\n2)File\n";

    cin >> option;

    cin.ignore();

    cout << "Enter key: ";

    getline(cin, key);

    strcpy(keyarr, key.c\_str());

    switch (option)

    {

    case 1:

        cout << "Enter text: ";

        break;

    case 2:

        freopen("input.txt", "r", stdin);

        freopen("output.txt", "w", stdout);

        break;

    }

    getline(cin, text);

    strcpy(textarr, text.c\_str());

    cipherText = PlayfairEncrypt(textarr, keyarr);

    cout << "Cipher Text: " << cipherText << endl;

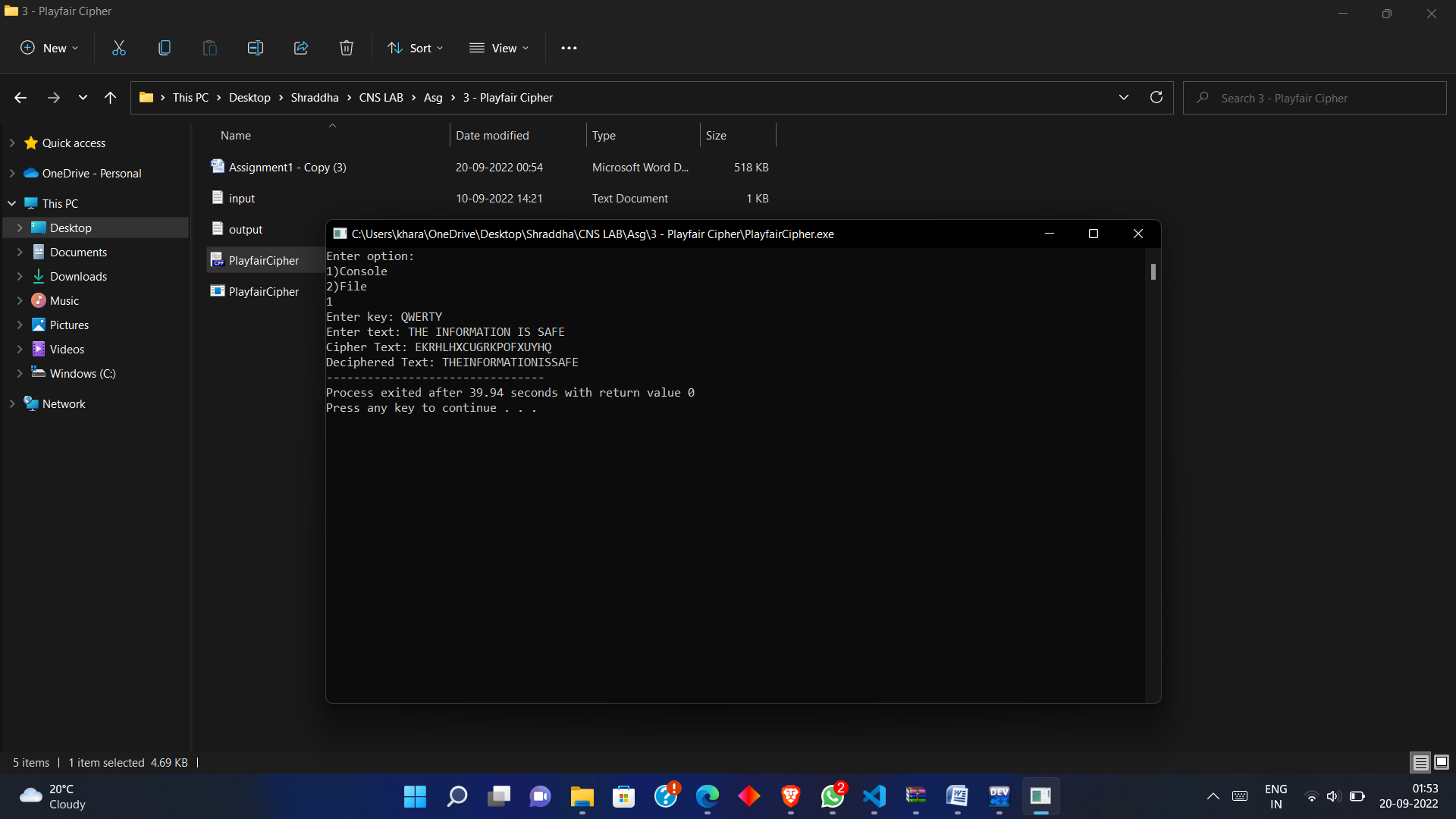
    strcpy(cipher, cipherText.c\_str());

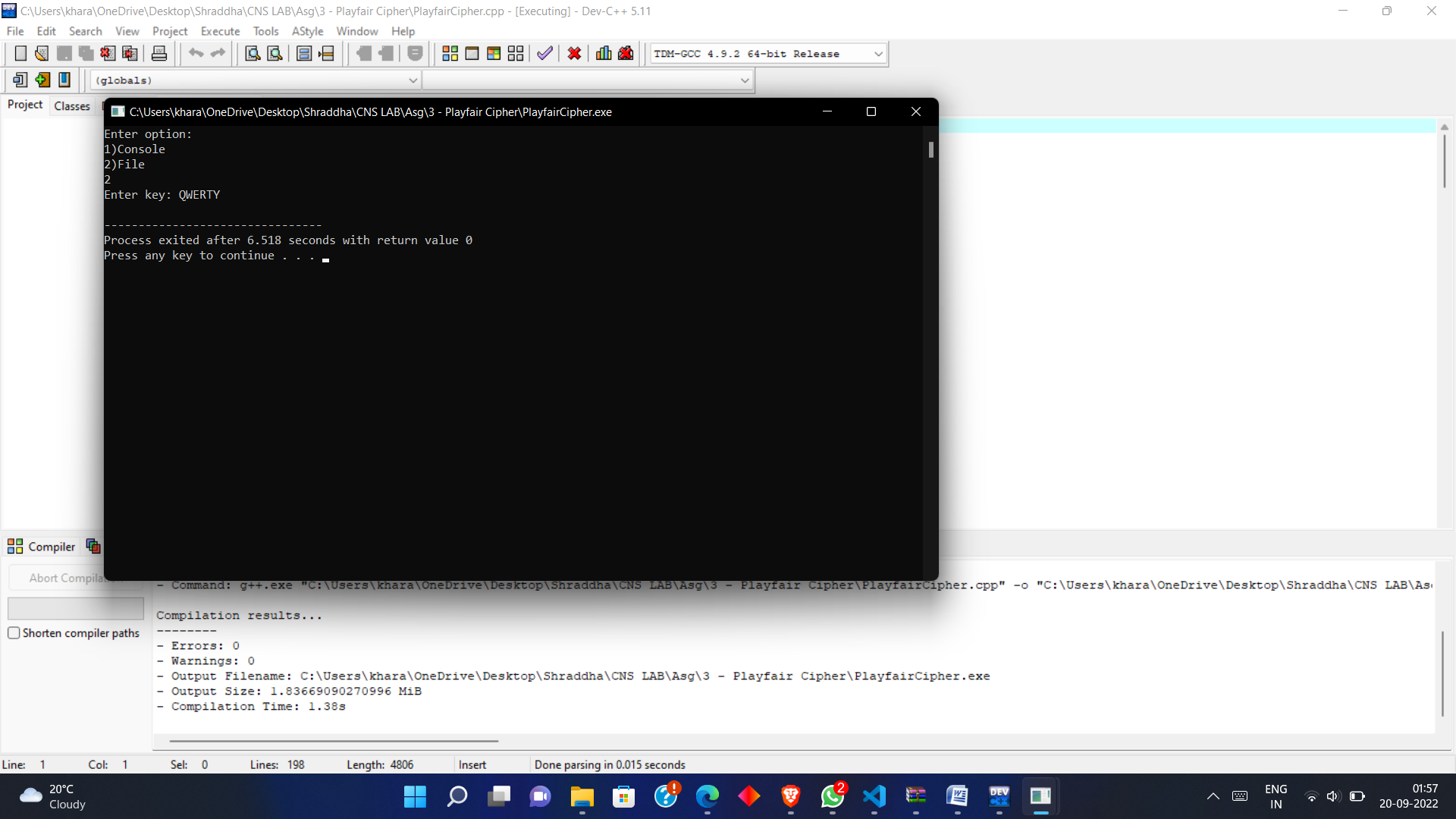
    cout << "Deciphered Text: " << PlayfairDecrypt(cipher, keyarr);

    return 0;

}

**Output:**





Cipher Text: UYEBEKREXCPY

Deciphered Text: SAVETHEWORLD

* **Conclusion** :

Play Fair is better algorithm than Caesar-cipher in security. But the PlayFair can be decrypted if the key is known.