# Lab - 03

Aim :- Write a program to implement
1. Autokey Cipher
2. playfair Cipher

Program:-1 : Autokey Cipher

Source code:-

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

string GenerateKey(int k1, string plainText)
{
    string key = "";
    key += (char)('a' + k1);
    for(int i=0; i<plainText.length(); ++i)
        key += plainText[i];
    return key;
}

String Encryption(string key, string plainText)
{
    string cipherText = "";
    for(int i=0; i<plainText.length(); ++i)
    {
        cipherText += ((plainText[i]-'a') +
                       (key[i]-'a')) %26;
        cipherText[i] = cipherText[i] + 'a';
    }
```

```
string Decryption (string key, string cipherText)
{
    string plainText = " ";
    for(int i=0; i<cipherText.length(); ++i)
    {
        plainText += ((cipherText[i]-'a') -
                      (key[i]-'a'));

        if(plainText[i] < 0)
            plainText[i] += 26;
        plainText[i] %= 26;
        plainText[i] = plainText[i] + 'a';
    }
    return plainText;
}
int main()
{
    int K1;
    string plainText;
    cin>> plainText >> K1;
    string key = GenerateKey(K1, plainText);
    string cipherText = Encryption( key, plainText);
    cout <<"encrypted Text: " << cipherText <<endl;
    string decrypted Text = Decryption( key, cipherText);
    cout <<" decrypted Text: " << decryptedText <<endl;
    return 0;
}
```

| Input | output |
| --- | --- |
| muneetdivan | Encrypted text :- Pshnfildvn |
| 3 | Decrypted text : muneetdivan |

Program-2 :- playfair Cipher

Source code:-

```
#include<bits/stdc++.h>
using namespace std;
#define N 5

void Generatekey(string kl, char key[N][N])
{
    Pair<int, int>
    Pair<char, bool> alphabets[26];
    for(int i=0; i<26; ++i)
    {
        alphabets[i] = make_pair('a'+i, true);
    }
    int k=0, r=0;
    for(int i=0; i<N; ++i)
    {
        for(int j=0; j<N; ++j)
        {
            if(k<kl.length())
            {
                key[i][j] = kl[k];
```

```
int m = k1[k] - 'a';
if( k1[k]=='i' || k1[k]=='j')
{
    alphabets[9].second = false;
    alphabets[8].second = false;
}
alphabets[m].second = false;
k++;
}
else
{
if(r<26 && alphabets[r].second)
{
    key[i][j] = alphabets[r].first;
    alphabets[r].second = false;
    if(r==8)
    {
        alphabets[++r].second = false;
    }
}
else
{
    j--;
}
}
}
}
}
```

```
pair<int, int> search( char ch, char key[N][N])
{
    if(ch == 'j')
        ch = 'i';
    pair<int,int> index;
    for(int i=0; i<N; ++i)
     for(int j = 0; j<N; ++j)
     {
         if( key[i][j] == ch)
             index = make_pair(i,j);
     }

    return index;
}

String Encryption( char key[N][N], string plainText)
{
    string cipherText = "";
    pair<int, int> index1, index2;
    for(int i = 0 ; i< plainText.length(); i+=2)
    {
        index1 = search(plainText[i], key);
        index2 = search( plainText[i+1], key);

        if( index1.first == index2.first)
        {
            index1.second = (index1.second +1) % N;
            index2.second = (index2.second +1) % N;
```

```
        ciphertext += key[index1.first][index1.second];
        ciphertext += key[index2.first][index2.second];
    }
    else if (index1.second == index2.second)
    {
        index1.first = (index1.first+1) % N;
        index2.first = (index2.first+1) % N;
        ciphertext += key[index1.first][index1.second];
        ciphertext += key[index2.first][index2.second];
    }
    else
    {
        ciphertext += key[index1.first][index1.second];
        ciphertext += key[index2.first][index2.second];
    }
  }
  return ciphertext;
}

String Decryption(char key[N][N], string ciphertext)
{
    String plaintext = "";
    pair<int,int> index1, index2;
    for(int i=0; i<ciphertext.length(); i+=2)
    {
        index1 = search(ciphertext[i], key);
        index2 = search(ciphertext[i], key);
```

```
if( index1.first == index2.first)
{
    index1.second--;
    index2.second--;
    if(index1.second<0)
        index1.second+=N;
    if(index2.second<0)
        index2.second+=N;
    index1.second %= N;
    index2.second %= N;
    plainText += key[index1.first][index1.second];
    plainText += key[index2.first][index2.second];
}
else if( index1.second == index2.second)
{
    index1.first--;
    index2.first--;
    if(index1.first<0)
        index1.first += N;
    if(index2.first<0)
        index2.first += N;
    index1.first %= N;
    index2.first %= N;
    plainText += key[index1.first][index1.second];
    plainText += key[index2.first][index2.second];
}
else
{
    plaintext += key[index1.first][index2.second];
    plainText += key[index2.first][index1.second];
}
```

```
    }
    return plainText;
}
int main()
{
    string k1, pt;
    cout<<"Enter key Text: ";
    getline(cin, k1);
    cout<<"enter your msg: ";
    cin>>pt;
    char key[N][N];
    Generate Key(k1, key);
    string cipherText = Encryption(key, pt);
    cout<<"encrypted text: "<<cipherText<<endl;
    string decrypted text = Decryption(key, cipherText);
    cout<<"Decrypted Text: "<<decryptedText<<endl;
    return 0;
}
```

| input | output |
| --- | --- |
| Enter key text:- munaf<br>enter your msg:- munafdivan | Encrypted text:- unafnghwtca<br>Decrypted text:- munafdivan. |