**Assignment – 6**

**Q1. Write a program to encrypt and decrypt the message "Meet Me at the Bridge" using Play fair cipher where key is "Your Name".**

**A:**

*Pseudocode and Explanation –*

1. **generating\_key\_matrix(string key)**

* **Purpose**: Creates a 5x5 matrix for the Playfair cipher using the given key. It avoids duplicate letters and excludes 'j'.
* **How it works**: The function first adds unique letters from the key to the matrix, then fills it with the rest of the alphabet (excluding 'j'), ensuring no letter repeats.

**2. add\_x(string plain\_text)**

* **Purpose**: Ensures the plaintext is ready for encryption by inserting 'x' between repeated letters and making the length even.
* **How it works**: It checks pairs of characters, inserts 'x' if two consecutive characters are the same, and appends 'x' if the final string has an odd length.

**3. get\_index(vector<vector<char>> matrix, char c)**

* **Purpose**: Finds the row and column of character c in the matrix.
* **How it works**: Iterates through the matrix to locate the character and returns its position as a pair of integers.

**4. main()**

* **Purpose**: Performs encryption and decryption using the Playfair cipher.
* **How it works**:
  + **Key Matrix Generation**: Calls generating\_key\_matrix to create the matrix.
  + **Plaintext Preprocessing**: Prepares the plaintext using add\_x.
  + **Encryption**: For each pair of characters, follows Playfair cipher rules (same row, same column, or rectangle) to generate the ciphertext.
  + **Decryption**: Reverses the encryption process to obtain the original text.

*Code -*

#include<bits/stdc++.h>

using *namespace* std;

vector<vector<*char*>> generating\_key\_matrix(string *key*){

    unordered\_map<*char*, *bool*> check\_occurence;

    vector<vector<*char*>> matrix;

*int* p = 0;

    vector<*char*> temp;

    for(*int* i = 0; i < *key*.length(); i++){

        if(!check\_occurence[*key*[i]]){

            check\_occurence[*key*[i]] = true;

            temp.push\_back(*key*[i]);

            p++;

            if(p % 5 == 0){

                matrix.push\_back(temp);

                temp.clear();

            }

        }

    }

    for(*int* i = 0; i < 26; i++){

        if(i == 9){

            continue;

        }

*char* current\_char = *char*(i + 97);

        if(!check\_occurence[current\_char]){

            temp.push\_back(current\_char);

            p++;

            if(p % 5 == 0){

                matrix.push\_back(temp);

                temp.clear();

            }

        }

    }

    if(!temp.empty()){

        matrix.push\_back(temp);

    }

    return matrix;

}

string add\_x(string *plain\_text*){

*int* i=0;

    string x = "x";

    while(i!=*plain\_text*.length()){

*char* p1 = *plain\_text*[i];

*char* p2;

        if((i+1) != *plain\_text*.length()){

            p2 = *plain\_text*[i+1];

        }else{

*plain\_text*.insert(i+1,x);

        }

        if(p1 == p2){

*plain\_text*.insert(i+1,x);

            i=0;

        }else{

            i+=2;

        }

    }

    return *plain\_text*;

}

pair<*int*,*int*> get\_index(vector<vector<*char*>> *matrix*, *char* *c*){

    for(*int* i=0;i<*matrix*.size();i++){

        for(*int* j=0;j<*matrix*[i].size();j++){

            if(*c* == *matrix*[i][j]){

                pair<*int*,*int*> p = {i,j};

                return p;

            }

        }

    }

}

*int* main()

{

    string key = "arin";

    vector<vector<*char*>> matrix = generating\_key\_matrix(key);

    for(*int* i = 0; i < matrix.size(); i++){

        for(*int* j = 0; j < matrix[i].size(); j++){

            cout << matrix[i][j] << " ";

        }

        cout << endl;

    }

    cout<<endl;

    string plain\_text = "meetmeatthebridge";

    string new\_plain\_text = add\_x(plain\_text);

    // cout<<new\_plain\_text<<endl;

*int* i=0;

    string cipher\_text;

    while(i!= new\_plain\_text.length()){

*char* c1 = new\_plain\_text[i];

*char* c2 = new\_plain\_text[i+1];

        pair<*int*,*int*> p1 = get\_index(matrix, c1);

        pair<*int*,*int*> p2 = get\_index(matrix, c2);

*int* i1 = p1.first;

*int* j1 = p1.second;

*int* i2 = p2.first;

*int* j2 = p2.second;

        // cout<<i1<<" "<<j1<<" "<<i2<<" "<<j2<<endl;

*char* new\_c1;

*char* new\_c2;

        if(i1 == i2){

            j1 = (j1+1)%5;

            j2 = (j2+1)%5;

            new\_c1 = matrix[i1][j1];

            new\_c2 = matrix[i2][j2];

        }else if(j1 == j2){

            i1 = (i1+1)%5;

            i2 = (i2+1)%5;

            new\_c1 = matrix[i1][j1];

            new\_c2 = matrix[i2][j2];

        }else{

            new\_c1 = matrix[i1][j2];

            new\_c2 = matrix[i2][j1];

        }

        cipher\_text.push\_back(new\_c1);

        cipher\_text.push\_back(new\_c2);

        i+=2;

    }

    cout<<"Cipher Text: "<<cipher\_text<<endl;

    string decrypted\_text;

*int* j=0;

    while(j!= cipher\_text.length()){

*char* c1 = cipher\_text[j];

*char* c2 = cipher\_text[j+1];

        pair<*int*,*int*> p1 = get\_index(matrix, c1);

        pair<*int*,*int*> p2 = get\_index(matrix, c2);

*int* i1 = p1.first;

*int* j1 = p1.second;

*int* i2 = p2.first;

*int* j2 = p2.second;

        // cout<<i1<<" "<<j1<<" "<<i2<<" "<<j2<<endl;

*char* new\_c1;

*char* new\_c2;

        if(i1 == i2){

            j1 = (j1-1)%5;

            j2 = (j2-1)%5;

            new\_c1 = matrix[i1][j1];

            new\_c2 = matrix[i2][j2];

        }else if(j1 == j2){

            i1 = (i1-1)%5;

            i2 = (i2-1)%5;

            new\_c1 = matrix[i1][j1];

            new\_c2 = matrix[i2][j2];

        }else{

            new\_c1 = matrix[i1][j2];

            new\_c2 = matrix[i2][j1];

        }

        decrypted\_text.push\_back(new\_c1);

        decrypted\_text.push\_back(new\_c2);

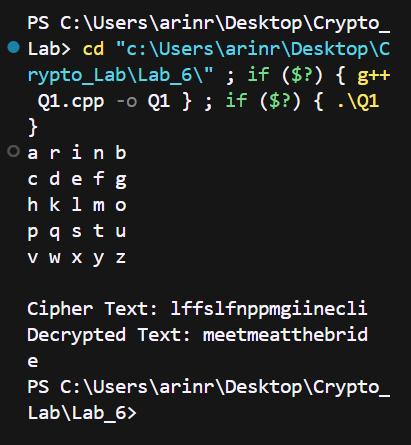
        j+=2;

    }

    cout<<"Decrypted Text: "<<decrypted\_text<<endl;

}

*Output –*

**

**Q2. Write a program to encrypt and decrypt the message "Pay more money" using trigraph Hill Cipher where key is "GYBNQKURP".**

**A:**

*Pseudocode and Explanation –*

1 **Key Matrix**:

* The key string ("rrfvsvcct") is converted into a 3x3 matrix by subtracting 97 from each character's ASCII value (to represent the letters' positions in the alphabet).

2 **Plaintext Padding**:

* The plaintext ("paymoremoney") is padded with 'x' characters so that its length becomes divisible by 3.

3 **Encryption**:

* For each 3-character block in the plaintext:
  + It is treated as a vector and multiplied by the key matrix.
  + The result is taken modulo 26 (to keep it within the alphabet range), and the corresponding characters are generated for the ciphertext.

4 **Decryption**:

* **Determinant and Inverse Calculation**:
  + The determinant of the key matrix is calculated using the determinant function.
  + The modular inverse of the determinant modulo 26 is found using the find\_MI function.
* **Cofactor and Inverse Matrix**:
  + The cofactors of the key matrix are calculated and transposed to find the adjugate matrix.
  + This adjugate matrix is multiplied by the modular inverse of the determinant and reduced modulo 26 to produce the inverse matrix.

5 **Decryption Process**:

* The ciphertext is decrypted by multiplying the inverse key matrix with each block of the ciphertext (same process as encryption, but using the inverse key matrix).

*Code –*

#include<bits/stdc++.h>

using *namespace* std;

*int* determinant(*int* *n* , vector<vector<*int*>> *a*) {

*int* det = (*a*[0][0]\*(*a*[1][1]\**a*[2][2] - *a*[1][2]\**a*[2][1])) - (*a*[0][1]\*(*a*[1][0]\**a*[2][2] - *a*[2][0]\**a*[1][2])) + (*a*[0][2]\*(*a*[1][0]\**a*[2][1] - *a*[2][0]\**a*[1][1]));

    return det;

}

*int* find\_MI(*int* *x1*, *int* *x2*){

*bool* x = false;

*int* c = 1;

        while(x != true){

            if((*x1*\*c)%*x2* == 1){

                x = true;

            }else{

                c++;

            }

        }

    return c;

}

*int* main()

{

    string key = "gybnqkurp";

    // string key = "rrfvsvcct";

    vector<vector<*int*>> key\_matrix;

*int* p=0;

    vector<*int*> temp;

    for(*int* i=0;i<key.length();i++){

        temp.push\_back(*int*(key[i])-97);

        p++;

        if(p%3 == 0){

            key\_matrix.push\_back(temp);

            temp.clear();

        }

    }

    // for(int i=0;i<key\_matrix.size();i++){

    //     for(int j=0;j<key\_matrix[i].size();j++){

    //         cout<<key\_matrix[i][j]<<" ";

    //     }

    //     cout<<endl;

    // }

    string plain\_text = "paymoremoney";

    if((plain\_text.length())%3 == 1){

        plain\_text.push\_back('x');

        plain\_text.push\_back('x');

    }else if((plain\_text.length())%3 == 2){

        plain\_text.push\_back('x');

    }

    // cout<<plain\_text;

*int* i=0;

    string cipher\_text;

    while(i!=plain\_text.length()){

*int* c1 = *int*(plain\_text[i]-97);

*int* c2 = *int*(plain\_text[i+1]-97);

*int* c3 = *int*(plain\_text[i+2]-97);

        for(*int* j=0;j<3;j++){

*int* val = c1\*key\_matrix[0][j] + c2\*key\_matrix[1][j] + c3\*key\_matrix[2][j];

*int* val2 = (val%26) +97;

            cipher\_text.push\_back(*char*(val2));

        }

        i+=3;

    }

    cout<<"Cipher Text: "<<cipher\_text<<endl;

    // Decryption

*int* n = key\_matrix.size();

*int* det = determinant(n, key\_matrix);

    if(det < 0){

*int* x = det\*-1;

        det = 26 - (x%26);

    }

*int* det\_inv = find\_MI(det,26);

    vector<vector<*int*>> cof;

    for(*int* i=0;i<n;i++){

        vector<*int*> temp;

        for(*int* j=0;j<n;j++){

*int* x = i;

*int* y = j;

*int* x1,x2;

*int* y1,y2;

            if(x == 0){

                x1 = 1;

                x2 = 2;

            }else if(x == 1){

                x1 = 0;

                x2 = 2;

            }else{

                x1 = 0;

                x2 = 1;

            }

            if(y == 0){

                y1 = 1;

                y2 = 2;

            }else if(y == 1){

                y1 = 0;

                y2 = 2;

            }else{

                y1 = 0;

                y2 = 1;

            }

*int* val = key\_matrix[x1][y1]\*key\_matrix[x2][y2] - key\_matrix[x1][y2]\*key\_matrix[x2][y1];

            if((x+y)%2 != 0){

                val = val\*-1;

            }

            temp.push\_back(val);

        }

        cof.push\_back(temp);

    }

    // Transpose matrix

    vector<vector<*int*>> k\_inverse;

    for(*int* i=0;i<n;i++){

        vector<*int*> temp;

        for(*int* j=0;j<n;j++){

            temp.push\_back(cof[j][i]);

        }

        k\_inverse.push\_back(temp);

    }

    for(*int* i=0;i<n;i++){

        for(*int* j=0;j<n;j++){

*int* fg = k\_inverse[i][j];

            if(fg < 0){

*int* x = fg\*-1;

                fg = 26 - (x%26);

                k\_inverse[i][j] = fg;

            }else{

                k\_inverse[i][j] = (k\_inverse[i][j])%26;

            }

        }

    }

    for(*int* i=0;i<n;i++){

        for(*int* j=0;j<n;j++){

*int* fg = k\_inverse[i][j]\*det\_inv;

            k\_inverse[i][j] = fg%26;

        }

        cout<<endl;

    }

    for(*int* i=0;i<n;i++){

        for(*int* j=0;j<n;j++){

            cout<<k\_inverse[i][j]<<" ";

        }

        cout<<endl;

    }

    string decrypted\_text="";

    i=0;

    while(i!=cipher\_text.length()){

*int* c1 = *int*(cipher\_text[i]-97);

*int* c2 = *int*(cipher\_text[i+1]-97);

*int* c3 = *int*(cipher\_text[i+2]-97);

        for(*int* j=0;j<3;j++){

*int* val = c1\*k\_inverse[0][j] + c2\*k\_inverse[1][j] + c3\*k\_inverse[2][j];

*int* val2 = (val%26) +97;

            // cout<<val2<<" ";

            decrypted\_text.push\_back(*char*(val2));

        }

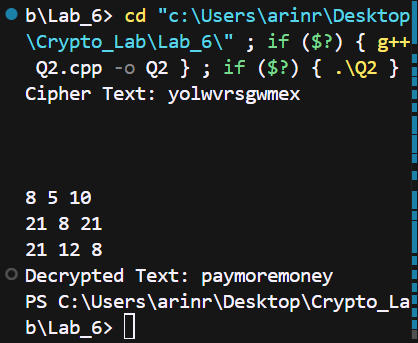
        i+=3;

    }

    cout<<"Decrypted Text: "<<decrypted\_text;

}

*Output –*

**