Nama : Andre Nathaniel Adipraja
NPM : 140810200042
Kelas : B

Hill Cipher

```
/*
    Nama       : Andre Nathaniel Adipraja
    NPM        : 140810200042
    Tanggal    : 27 September 2022
    Deskripsi : Program C++ Hill Cipher
*/

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

int key[3][3] ; // Sebagai kunci global batas ordo matriks

int mod26(int x){ // fungsi untuk mod

    return x >= 0 ? (x%26) : 26-(abs(x)%26) ;
}

// Mencari determinan matriks
int cariDeterminan(int m[3][3] , int n){
    int det;
    if(n == 2){ // jika ordo matriks = 2

        det = m[0][0] * m[1][1] - m[0][1] * m[1][0] ;
    }
    else det = 0 ; // invalid input
    return mod26(det);
}

int cariDeterminanInverse(int R , int D = 26){ // Mencari invers matriks

// R adalah sisa atau determinan
    int i = 0 ;
    int p[100] = {0, 1};
    int q[100] = {0} ; // hasil bagi
```

```
    while(R != 0){ // jika sisa tidak sama dengan 0

        q[i] = D / R ;
        int oldD = D ;
        D = R;
        R = oldD % R ;
        if(i > 1)
        {
            p[i] = mod26(p[i - 2] - p[i - 1] * q[i - 2]) ;
        }
        i++ ;
    }
    if (i == 1) return 1;
    else return p[i] = mod26(p[i - 2] - p[i - 1] * q[i - 2]) ;
}
int gcd(int m, int n){
    if (n > m)
        swap(m, n);

    do{
        int temp = m % n;
        m = n;
        n = temp;
    } while (n != 0);

    return m;
}

void perkalianMatrix(int a[1000][3] , int a_baris , int a_kolom ,  int
b[1000][3] , int b_baris , int b_kolom , int res[1000][3]){
    for(int i = 0 ; i < a_baris ; i++){
        for(int j = 0 ; j < b_kolom ; j++){
            for(int k = 0 ; k < b_baris ; k++){
                res[i][j] += a[i][k] * b[k][j] ;
            }
            res[i][j] = mod26(res[i][j]) ;
        }
    }
}
void cariKunci(){
```

```
    //deklarasi
    string plainteks,cipherteks;
    int key[2][2], det, detInv, adj[2][2], plainteksInv[2][2],
plainMatrix[2][2], cipMatrix[2][2], counter;
    int p, c;
    int transpose[2][2];

    //input plainteks
    cout << "Masukkan Plain text : ";
    cin.ignore();
    getline(cin, plainteks);

    //assign plainteks ke plainMatrix
    counter = 0;
    for(int i = 0; i < 2; i++){
        for(int j = 0; j < 2; j++){
            p = toupper(plainteks[counter]) - 65;
            plainMatrix[i][j] = p;
            counter++;
        }
    }

    //input cipherteks
    cout << "Masukkan Cipher text : ";
    getline(cin, cipherteks);

    //assign cipherteks ke cipMatrix
    counter = 0;
    for(int i = 0; i < 2; i++){
        for(int j = 0; j < 2; j++){
            c = toupper(cipherteks[counter]) - 65;
            cipMatrix[i][j] = c;
            counter++;
        }
    }

    // determinan
    det = (plainMatrix[0][0] * plainMatrix[1][1]) - (plainMatrix[0][1] *
plainMatrix[1][0]);
    if(gcd(det, 26) == 1){
```

```
        // inverse dari determinan mod 26
        detInv = cariDeterminanInverse(det, 26);

        //menghitung adjoin
        adj[0][0] = plainMatrix[1][1];
        adj[0][1] = (-1) * plainMatrix[0][1];
        adj[1][0] = (-1) * plainMatrix[1][0];
        adj[1][1] = plainMatrix[0][0];

        //menghitung matriks invers dari plainteks
        for(int i = 0; i < 2; i++){
            for(int j = 0; j < 2; j++){
                plainteksInv[i][j] = detInv * adj[i][j];
                if(plainteksInv[i][j] < 0){
                    plainteksInv[i][j] = 26 - (abs(plainteksInv[i][j]) %
26);
                }else{
                    plainteksInv[i][j] = plainteksInv[i][j];
                    plainteksInv[i][j] = plainteksInv[i][j] % 26;
                }
            }
        }

        //Search key
        for(int i = 0; i < 2; i++){
            for(int j = 0; j < 2; j++){
                key [i][j] = 0;
                for(int k = 0; k < 2; k++){
                    key [i][j] += (plainteksInv[i][k] * cipMatrix[k][j]);
                }
                key [i][j] %= 26;
            }
        }


        for (int i = 0; i < 2; i++){
        for (int j = 0; j < 2; j++){
            transpose[j][i] = key[i][j];
            }
        }
```

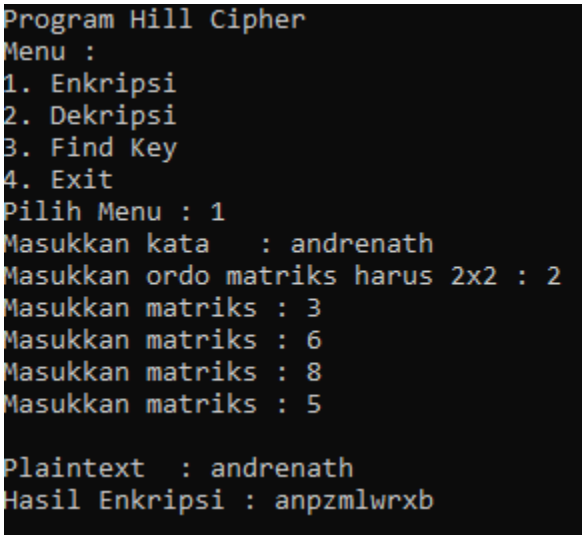
```
        for(int i = 0; i < 2; i++){
            for (int j = 0; j < 2; j++){
                cout << (transpose[i][j]) << "\t";
            }
            cout <<endl;
        }
    }else{
        cout << "Determinan tidak relatif" << endl;
        cout << "Kunci tidak ditemukan" << endl << endl;
    }

}
/* Invers = (matriks * det Invers) mod 26 */
/* cari Invers(matrix , order_of_matrix , result_matrix) */
void cariInvers(int m[3][3] , int n , int m_inverse[3][3] ){
    int adj[3][3] = {0};

    int det = cariDeterminan(m , n); // ini menggunakan fungsi
cariDeterminan(matrix , order_of_matrix)
    int detInverse = cariDeterminanInverse(det);

    if(n==2){ //jika ordo matrik 2x2
        adj[0][0] = m[1][1];
        adj[1][1] = m[0][0];
        adj[0][1] = -m[0][1];
        adj[1][0] = -m[1][0];
    }

    for(int i = 0; i < n ; i++){
        for(int j = 0; j < n; j++){
            m_inverse[i][j] = mod26(adj[i][j] * detInverse) ;
        }
    }
}

// C = PK
string encrypt(string pt, int n){
    int P[1000][3] = {0} ; // plaintext
    int C[1000][3] = {0} ; // cipher text
```

```
    int ptIter = 0  ;

    while(pt.length() % n != 0){
        pt += "x" ;  // ini digunakan jika plaintext di module dengan
matrik tidak sama dengan 0
    }
    int row = (pt.length())/n; // jumlah baris dalam plaintext

    for(int i = 0; i < row ; i++){
        for(int j = 0; j < n; j++){
            P[i][j] = pt[ptIter++]-'a' ;
        }
    }

    // perkalianMatrix(mat_a , row_a , col_a , mat_b, row_b, col_b ,
mat_result)
    perkalianMatrix(P, row , n , key , n , n , C) ;

    string ct = "" ;
    for(int i = 0 ; i < row ; i++){
        for(int j = 0 ; j < n ; j++){
            ct += (C[i][j] + 'a');
        }
    }
    return ct ;
}

// P = C*(k_inverse)
string decrypt(string ct, int n){
    int P[1000][3] = {0} ; // plaintext
    int C[1000][3] = {0} ; // cipher text
    int ctIter = 0 ;

    int row = ct.length()/n; // banyak baris di chipertext

    for(int i = 0; i < row ; i++){
        for(int j = 0; j < n; j++){
            C[i][j] = ct[ctIter++]-'a' ;
        }
    }
```

```
    int k_inverse[3][3] = {0};
    /* cariInvers(matrix , order_of_matrix , result_matrix) */
    cariInvers(key, n , k_inverse);

    /* perkalianMatrix(mat_a , row_a , col_a , mat_b, row_b, col_b ,
mat_result) */
    perkalianMatrix(C, row , n , k_inverse , n , n , P) ;

    string pt = "" ;
    for(int i = 0 ; i < row ; i++){
        for(int j = 0 ; j < n ; j++){
            pt += (P[i][j] + 'a');
        }
    }
    return pt ;
}

int main(void){
    bool menuActive = true;
    string pt, ct;
    int n ;
    int pilih;

    while(menuActive){
        cout << "\nProgram Hill Cipher" <<endl;
        cout << "Menu : " <<endl;
        cout << "1. Enkripsi" <<endl;
        cout << "2. Dekripsi" <<endl;
        cout << "3. Find Key" <<endl;
        cout << "4. Exit" <<endl;
        cout << "Pilih Menu : ";
            cin >> pilih;
        switch(pilih){
            case 1:
                    cout << "Masukkan kata   : " ;
                        cin >> pt;

                    cout << "Masukkan ordo matriks harus 2x2 : ";
                        cin >> n ;
```

```
                    for(int i=0; i<n; i++){
                        for(int j=0; j<n; j++){
                            cout<<"Masukkan matriks : ";
                                cin >> key[i][j];
                        }
                    }

                    cout << "\nPlaintext  : " << pt << endl;

                    ct = encrypt(pt, n) ;
                    cout << "Hasil Enkripsi : " << ct << endl;
                    break;
            case 2:
                    cout << "Masukkan kata   : " ;
                        cin >> ct;

                    cout << "Masukkan ordo matriks harus 2x2 : ";
                        cin >> n ;

                    for(int i=0; i<n; i++){
                        for(int j=0; j<n; j++){
                            cout<<"Masukkan matriks : ";
                                cin >> key[i][j];
                        }
                    }

                    cout << "\nChipertext  : " << ct << endl;
                    cout << "Hasil Dekripsi : " << decrypt(ct, n)<< endl;
                    break;
            case 3:
                    cout<<endl;
                    cariKunci();
                    break;
            default:
                    cout << "\nInvalid Input" <<endl;
                    break;
        }
    }
}
```

Screenshot program :



Penjelasan :

1. Fungsi cariDeterminan untuk mencari determinan dari matriks, dicek terlebih dahulu jika matriks ordo 2x2 maka akan dihitung, jika != 2 maka akan invalid.
2. Fungsi cariDeterminanInverse untuk mencari inverse dari determinan matriks tersebut, dicek apakah R yang merupakan determinan != 0 jika iya maka akan di modulo dengan 26.
3. Fungsi mod untuk memodulokan rumusnya.
4. Fungsi cariInvers untuk mencari inverse matrik, menggunakan fungsi determinan dan inverse determinan lalu dicek ordo jika 2x2 maka akan dibentuk adjoin dari matrik tersebut. Lalu setelah itu akan di inverse pada rumus m_inverse[i][j] = mod26(adj[i][j] * detInverse) ;
5. Fungsi enkripsi untuk mengenkripsi plaintext, dicek dahulu apakah huruf%ordo matriks == 0 jika iya maka langsung diubah menjadi matriks dan dikalikan oleh key, jika kurang akan ditambahkan x pada bagian belakang plaintext. setelah itu di enkripsi dan ditambahkan ‘a’ agar sesuai dengan Ascii.
6. Fungsi dekripsi sama dengan enkripsi hanya menggunakan inverse matriks.
7. Fungsi gcd untuk mengecek gcd harus = 1
8. Fungsi find key untuk menemukan kunci dari plaintext dicek dulu apakah gcd plaintext =1 jika tidak maka determinan tidak relatif kunci tidak dapat ditemukan.
9. Kekurangan pada program ini hanya dapat menggunakan matriks ber ordo 2 x 2 dan huruf wajib kecil semua