Tugas Kecil 1 IF4020 Kriptografi

Penyusunan Kriptografi Klasik dengan GUI (Crypt GUI)

Nama: Leonard Matheus

NIM: 13519215

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2021

Source Code Program C++

1. Vigenere Cipher Standard - VCS.cpp

```cpp
#include "VCS.h"
#include <iostream>
#include <string>

using namespace std;

VCS::VCS(){
    plain = "null";
    kunci = "null";
    cipher = "null";
    file = "null";
}

void VCS::VCS_Enkripsi()
{
    string plain2 = getPlain();
    //Remove karakter non-alphabetic
    string plain = removeKarakterLain(plain2);
    //Menyelaraskan kunci dengan panjang Plain text
    string kunci = generateKunci(plain, this->getKunci());
    string tampungan2 = "";
    for (int i = 0; i < (int) plain.size(); i++) {
        int p = charToInt(plain[i], true);
        int k = charToInt(kunci[i], true);
        int c = (p + k) % 26;
        char cnya = intToChar(c, true);
        tampungan2.push_back(cnya);
    }
    string temp = filterOutput(tampungan2, 5);
    setCipher(temp);
}
```

```cpp
void VCS::VCS_Enkripsi_File()
{
    string cipher = getCipher();
    setFile(cipher);
}

void VCS::VCS_Dekripsi() {
    string cipher2 = getCipher();
    //Remove karakter non-alphabetic
    string cipher = removeKarakterLain(cipher2);
    //Menyelaraskan kunci dengan panjang Plain text
    string kunci = generateKunci(cipher, this->getKunci());
    string tampungan2 = "";
    for (int i = 0; i < (int)cipher.size(); i++) {
        int c = charToInt(cipher[i], true);
        int k = charToInt(kunci[i], true);
        if (c - k < 0) {
            c += 26;
        }
        int p = (c - k) % 26;
        char pnya = intToChar(p, true);
        tampungan2.push_back(pnya);
    }
    setPlain(tampungan2);
}

void VCS::VCS_Dekripsi_File() {
    string plain = getPlain();
    setFile(plain);
}
```

```cpp
string VCS::generateKunci(string _plain, string _kunci)
{
    string tampungan;
    int jumlahPlain = (int) _plain.size();
    int jumlahKunci = (int) _kunci.size();
    int faktor = ceil(jumlahPlain / jumlahKunci);
    for (int i = 0; i <= faktor; i++) {
        tampungan += kunci;
    }
    tampungan.resize(jumlahPlain);
    return tampungan;
}

string VCS::getPlain(){return this->plain;}
string VCS::getKunci() { return this->kunci; }
string VCS::getCipher() { return this->cipher; }
string VCS::getFile() { return this->file; }

void VCS::setPlain(string _plain){this->plain = _plain;}
void VCS::setKunci(string _kunci) { this->kunci = removeKarakterLain(_kunci); }
void VCS::setCipher(string _cipher) { this->cipher = _cipher; }
void VCS::setFile(string _file) { this->file = _file; }
```

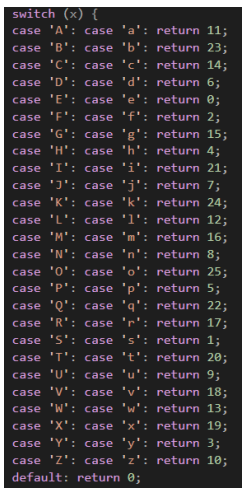
2. Full Vigenere Cipher - FCS.cpp

   Pergeseran key berdasarkan urutan alphabet yang telah di hard-code sehingga tinggal dihitung modulusnya. Hardcode key dapat ditemukan di backend.cpp

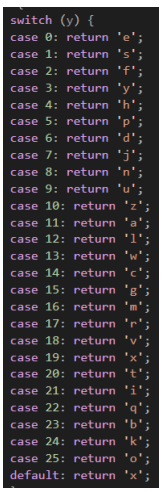
```cpp
#include "FVC.h"
#include <iostream>
#include <string>

using namespace std;

FVC::FVC() {
    plain = "null";
    kunci = "null";
    cipher = "null";
    file = "null";
}

void FVC::FVC_Enkripsi()
{
    string plain2 = getPlain();
    //Remove karakter non-alphabetic
    string plain = removeKarakterLain(plain2);
    //Menyelaraskan kunci dengan panjang Plain text
    string kunci = generateKunci(plain, this->getKunci());
    string tampungan2 = "";
    for (int i = 0; i < (int)plain.size(); i++) {
        int p = charToInt(plain[i], false);
        int k = charToInt(kunci[i], false);
        int c = (p + k) % 26;
        char cnya = intToChar(c,false);
        tampungan2.push_back(cnya);
    }
    string temp = filterOutput(tampungan2, 5);
    setCipher(temp);
}
```

```cpp
void FVC::FVC_Enkripsi_File()
{
    string cipher = getCipher();
    setFile(cipher);
}

void FVC::FVC_Dekripsi() {
    string cipher2 = getCipher();
    //Remove karakter non-alphabetic
    string cipher = removeKarakterLain(cipher2);
    //Menyelaraskan kunci dengan panjang Plain text
    string kunci = generateKunci(cipher, this->getKunci());
    string tampungan2 = "";
    for (int i = 0; i < (int)cipher.size(); i++) {
        int c = charToInt(cipher[i],false);
        int k = charToInt(kunci[i],false);
        if (c - k < 0) {
            c += 26;
        }
        int p = (c - k) % 26;
        //setCipherFile(static_cast<string>(to_string(p)));
        char pnya = intToChar(p,false);
        tampungan2.push_back(pnya);
    }
    setPlain(tampungan2);
}

void FVC::FVC_Dekripsi_File() {
    string plain = getPlain();
    setFile(plain);
}
```

```cpp
string FVC::generateKunci(string _plain, string _kunci)
{
    string tampungan;
    int jumlahPlain = (int)_plain.size();
    int jumlahKunci = (int)_kunci.size();
    int faktor = ceil(jumlahPlain / jumlahKunci);
    for (int i = 0; i <= faktor; i++) {
        tampungan += kunci;
    }
    tampungan.resize(jumlahPlain);
    return tampungan;
}

string FVC::getPlain() { return this->plain; }
string FVC::getKunci() { return this->kunci; }
string FVC::getCipher() { return this->cipher; }
string FVC::getFile() { return this->file; }

void FVC::setPlain(string _plain) { this->plain = _plain; }
void FVC::setKunci(string _kunci) { this->kunci = removeKarakterLain(_kunci); }
void FVC::setCipher(string _cipher) { this->cipher = _cipher; }
void FVC::setFile(string _file) { this->file = _file; }
```



```cpp
switch (x) {
case 'A': case 'a': return 11;
case 'B': case 'b': return 23;
case 'C': case 'c': return 14;
case 'D': case 'd': return 6;
case 'E': case 'e': return 0;
case 'F': case 'f': return 2;
case 'G': case 'g': return 15;
case 'H': case 'h': return 4;
case 'I': case 'i': return 21;
case 'J': case 'j': return 7;
case 'K': case 'k': return 24;
case 'L': case 'l': return 12;
case 'M': case 'm': return 16;
case 'N': case 'n': return 8;
case 'O': case 'o': return 25;
case 'P': case 'p': return 5;
case 'Q': case 'q': return 22;
case 'R': case 'r': return 17;
case 'S': case 's': return 1;
case 'T': case 't': return 20;
case 'U': case 'u': return 9;
case 'V': case 'v': return 18;
case 'W': case 'w': return 13;
case 'X': case 'x': return 19;
case 'Y': case 'y': return 3;
case 'Z': case 'z': return 10;
default: return 0;
```



```cpp
switch (y) {
case 0: return 'e';
case 1: return 's';
case 2: return 'f';
case 3: return 'y';
case 4: return 'h';
case 5: return 'p';
case 6: return 'd';
case 7: return 'j';
case 8: return 'n';
case 9: return 'u';
case 10: return 'z';
case 11: return 'a';
case 12: return 'l';
case 13: return 'w';
case 14: return 'c';
case 15: return 'g';
case 16: return 'm';
case 17: return 'r';
case 18: return 'v';
case 19: return 'x';
case 20: return 't';
case 21: return 'i';
case 22: return 'q';
case 23: return 'b';
case 24: return 'k';
case 25: return 'o';
default: return 'x';
```

3. Auto-Key Vigenere Cipher - AVC.cpp

```cpp
#include "AVC.h"
#include <iostream>
#include <string>

using namespace std;

AVC::AVC() {
        plain = "null";
        kunci = "null";
        cipher = "null";
        file = "null";
}

void AVC::AVC_Enkripsi()
```

```
{
        string plain2 = getPlain();
        //Remove karakter non-alphabetic
        string plain = removeKarakterLain(plain2);
        //Menyelaraskan kunci dengan panjang Plain text
        string kunci = generateKunci(plain, this->getKunci());
        string tampungan2 = "";
        for (int i = 0; i < (int)plain.size(); i++) {
                int p = charToInt(plain[i], true);
                int k = charToInt(kunci[i], true);
                int c = (p + k) % 26;
                char cnya = intToChar(c, true);
                tampungan2.push_back(cnya);
        }
        string temp = filterOutput(tampungan2, 5);
        setCipher(temp);
}

void AVC::AVC_Enkripsi_File()
{
        string cipher = getCipher();
        setFile(cipher);
}

void AVC::AVC_Dekripsi() {
        string cipher2 = getCipher();
        //Remove karakter non-alphabetic
        string cipher = removeKarakterLain(cipher2);
        //Menyelaraskan kunci dengan panjang Plain text
        string kunci = generateKunci(cipher, this->getKunci());
        string tampungan2 = "";
        for (int i = 0; i < (int)cipher.size(); i++) {
                int c = charToInt(cipher[i], true);
                int k = charToInt(kunci[i], true);
                if (c - k < 0) {
                        c += 26;
                }
                int p = (c - k) % 26;
                char pnya = intToChar(p, true);
                tampungan2.push_back(pnya);
        }
        setPlain(tampungan2);
}

void AVC::AVC_Dekripsi_File() {
        string plain = getPlain();
        setFile(plain);
}


string AVC::generateKunci(string _plain, string _kunci)
{
        string tampungan;
        tampungan += _kunci;
        tampungan += removeKarakterLain(_plain);
        tampungan.resize((int) _plain.size());
        return tampungan;
}

string AVC::getPlain() { return this->plain; }
string AVC::getKunci() { return this->kunci; }
string AVC::getCipher() { return this->cipher; }
string AVC::getFile() { return this->file; }

void AVC::setPlain(string _plain) { this->plain = _plain; }
void AVC::setKunci(string _kunci) { this->kunci = removeKarakterLain(_kunci); }
void AVC::setCipher(string _cipher) { this->cipher = _cipher; }
void AVC::setFile(string _file) { this->file = _file; }
```

4. Extended Vigenere Cipher - FCS.cpp

```cpp
#include "EVC.h"
#include <iostream>
#include <string>

using namespace std;

EVC::EVC() {
    plain = "null";
    kunci = "null";
    cipher = "null";
    file = "null";
}

void EVC::EVC_Enkripsi()
{
    string plain2 = getPlain();
    //Menyelaraskan kunci dengan panjang Plain text
    string kunci = generateKunci(plain, this->getKunci());
    string tampungan2 = "";
    for (int i = 0; i < (int)plain.size(); i++) {
        int p = charToIntEVC(plain[i]);
        int k = charToIntEVC(kunci[i]);
        int c = (p + k) % 256;
        char cnya = intToCharEVC(c);
        tampungan2.push_back(cnya);
    }
    //string temp = filterOutput(tampungan2, 5);
    setCipher(tampungan2);
}
```

```cpp
void EVC::EVC_Enkripsi_File()
{
    string cipher = getCipher();
    setFile(cipher);
}

void EVC::EVC_Dekripsi() {
    string cipher2 = getCipher();
    //Menyelaraskan kunci dengan panjang Plain text
    string kunci = generateKunci(cipher, this->getKunci());
    string tampungan2 = "";
    for (int i = 0; i < (int)cipher.size(); i++) {
        int c = charToIntEVC(cipher[i]);
        int k = charToIntEVC(kunci[i]);
        if (c - k < 0) {
            c += 256;
        }
        int p = (c - k) % 256;
        char pnya = intToCharEVC(p);
        tampungan2.push_back(pnya);
    }
    setPlain(tampungan2);
}

void EVC::EVC_Dekripsi_File() {
    string plain = getPlain();
    setFile(plain);
}
```

```cpp
string EVC::generateKunci(string _plain, string _kunci)
{
    string tampungan;
    int jumlahPlain = (int)_plain.size();
    int jumlahKunci = (int)_kunci.size();
    int faktor = ceil(jumlahPlain / jumlahKunci);
    for (int i = 0; i <= faktor; i++) {
        tampungan += kunci;
    }
    tampungan.resize(jumlahPlain);
    return tampungan;
}

string EVC::getPlain() { return this->plain; }
string EVC::getKunci() { return this->kunci; }
string EVC::getCipher() { return this->cipher; }
string EVC::getFile() { return this->file; }

void EVC::setPlain(string _plain) { this->plain = _plain; }
void EVC::setKunci(string _kunci) { this->kunci = _kunci; }
void EVC::setCipher(string _cipher) { this->cipher = _cipher; }
void EVC::setFile(string _file) { this->file = _file; }
```

5. Playfair Cipher - PFCip.cpp

```cpp
#include "PFCip.h"
#include <iostream>
#include <string>
#include <algorithm>
#include <map>

using namespace std;

PFCip::PFCip() {
        plain = "null";
        kunci = "null";
        cipher = "null";
        file = "null";
}

int PFCip::getIndex(vector<char> v, int K)
{
```

```
        auto it = find(v.begin(), v.end(), K);
        if (it != v.end())
        {
                int index = it - v.begin();
                return index;
        }
        else {
                return -1;
        }
}

pair<int, int> PFCip::letakHuruf(vector<vector<char>> m, char c) {
        pair<int, int> index;
        for (unsigned int i = 0; i < m.size(); i++) {
                if (getIndex(m.at(i), c) != -1) {
                        index.second = getIndex(m.at(i), c);
                        index.first = i;
                        return index;
                }
        }
        //Kalau characternya tidak ada
        index.second = -1;
        index.first = -1;
        return index;
}

void PFCip::PFCip_Enkripsi()
{
        string _kunci = this->getKunci();
        vector<char> v, v2;
        vector<vector<char>> v3;
        std::copy(_kunci.begin(), _kunci.end(), std::back_inserter(v));
        v2 = unique_elements(v);
        v3 = matriksKey(v2);
        string plain2 = getPlain();
        //Remove karakter non-alphabetic
        string plain = removeKarakterLain(plain2);
        for (unsigned int j = 0; j < plain.size(); j++) {
                if (plain[j] == 'J' || plain[j] == 'j')
                {
                        plain.erase(j, 1);
                        j--;
                }
        }
        //setting plain text nya
        unsigned int i = 0;
        while ( i < plain.size()) {
                if (plain[i] == plain[i + 1]) {
                        plain.insert((i + 1), 1, 'X');
                }
                i += 2;
        }
        if (plain.size() % 2 == 1) {
                plain.push_back('X');
        }
        //Proses enkripsi
        string cipher;
        unsigned int k = 0;
        while (k < plain.size()) {
                pair<int, int> huruf1, huruf2;
                huruf1 = letakHuruf(v3, plain[k]);
                huruf2 = letakHuruf(v3, plain[k + 1]);
                //Barisnya sama
                if (huruf1.first == huruf2.first) {
                        huruf1.second += 1;
                        huruf1.second %= 5;
                        huruf2.second += 1;
                        huruf2.second %= 5;
```

```
                        cipher.append(1, v3.at(huruf1.first).at(huruf1.second));
                        cipher.append(1, v3.at(huruf2.first).at(huruf2.second));
                }
                //Kolomnya sama
                else if (huruf1.second == huruf2.second) {
                        huruf1.first += 1;
                        huruf1.first %= 5;
                        huruf2.first += 1;
                        huruf2.first %= 5;
                        cipher.append(1, v3.at(huruf1.first).at(huruf1.second));
                        cipher.append(1, v3.at(huruf2.first).at(huruf2.second));
                }
                else {
                        cipher.append(1, v3.at(huruf1.first).at(huruf2.second));
                        cipher.append(1, v3.at(huruf2.first).at(huruf1.second));
                }
                k += 2;
        }
        string temp = filterOutput(cipher, 5);
        setCipher(temp);
}

void PFCip::PFCip_Enkripsi_File()
{
        string cipher = getCipher();
        setFile(cipher);
}

void PFCip::PFCip_Dekripsi() {
        string _kunci = this->getKunci();
        vector<char> v, v2;
        vector<vector<char>> v3;
        std::copy(_kunci.begin(), _kunci.end(), std::back_inserter(v));
        v2 = unique_elements(v);
        v3 = matriksKey(v2);
        string cipher = getCipher();
        //Remove karakter non-alphabetic
        cipher = removeKarakterLain(cipher);
        for (unsigned int j = 0; j < cipher.size(); j++) {
                if (cipher[j] == 'J' || cipher[j] == 'j')
                {
                        cipher.erase(j, 1);
                        j--;
                }
        }
        //Proses dekripsi
        string plain;
        unsigned int k = 0;
        while (k < cipher.size()) {
                pair<int, int> huruf1, huruf2;
                huruf1 = letakHuruf(v3, cipher[k]);
                huruf2 = letakHuruf(v3, cipher[k + 1]);
                //Barisnya sama
                if (huruf1.first == huruf2.first) {
                        if ((huruf1.second) - 1 < 0) {
                                huruf1.second += 5;
                        }
                        huruf1.second -= 1;
                        if ((huruf2.second) - 1 < 0) {
                                huruf2.second += 5;
                        }
                        huruf2.second -= 1;
                        plain.append(1, v3.at(huruf1.first).at(huruf1.second));
                        plain.append(1, v3.at(huruf2.first).at(huruf2.second));
                }
                //Kolomnya sama
                else if (huruf1.second == huruf2.second) {
                        if ((huruf1.first) - 1 < 0) {
```

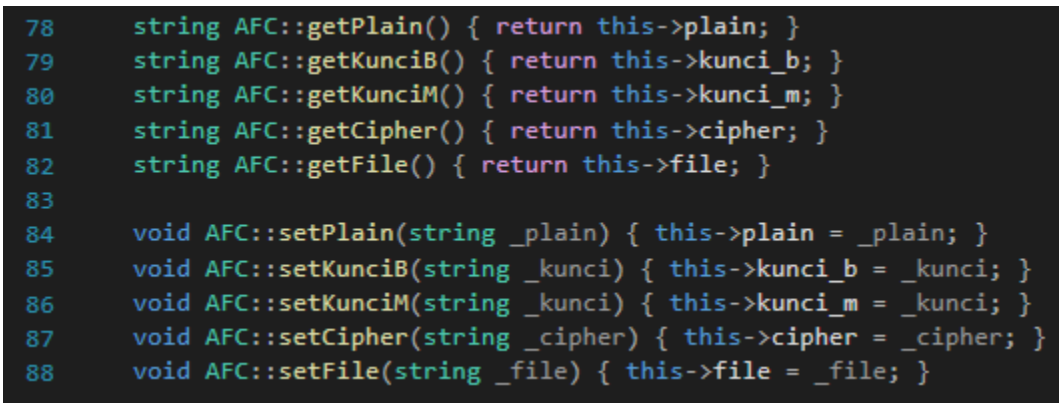
```
                                huruf1.first += 5;
                        }
                        huruf1.first -= 1;
                        if ((huruf2.first) - 1 < 0) {
                                huruf2.first += 5;
                        }
                        huruf2.first -= 1;
                        plain.append(1, v3.at(huruf1.first).at(huruf1.second));
                        plain.append(1, v3.at(huruf2.first).at(huruf2.second));
                }
                else {
                        plain.append(1, v3.at(huruf1.first).at(huruf2.second));
                        plain.append(1, v3.at(huruf2.first).at(huruf1.second));
                }
                k += 2;
        }
        //setting plain text nya
        for (unsigned int i = 0; i < plain.size(); i++) {
                if (plain[i] == 'X' || plain[i] == 'x')
                {
                        plain.erase(i, 1);
                        i--;
                }
        }
        setPlain(plain);
}

void PFCip::PFCip_Dekripsi_File() {
        string plain = getPlain();
        setFile(plain);
}

vector<char> PFCip::unique_elements(vector<char>& vec)
{
        vector<char> vec2;
        map<char, int> m;
        for (auto p = vec.begin(); p != vec.end(); ++p) {
                if (*p != 'J' && *p >= 'A' && *p <= 'Z') {
                        m[*p]++;
                }
                if (m[*p] > 0 && m[*p] < 2) {
                        vec2.push_back(*p);
                }
        }
        for (char i = 'A'; i <= 'Z'; i++) {
                if (i != 'J' && m[i] == 0) {
                        vec2.push_back(i);
                }
        }
        return vec2;
}

vector<vector<char>> PFCip::matriksKey(vector<char> kunci) {
        vector<vector<char>> m;
        for (int i = 0; i < 25; i++) {
                if (i % 5 == 0) {
                        m.push_back(vector<char>());
                }
                m.at(round(i / 5)).push_back(kunci.at(i));
        }
        return m;
}

vector<vector<char>> PFCip::generateKunci(string _kunci)
{
        vector<char> v, v2;
        vector<vector<char>> v3;
        std::copy(_kunci.begin(), _kunci.end(), std::back_inserter(v));
```

```
        v2 = unique_elements(v);
        return matriksKey(v2);
}

string PFCip::getPlain() { return this->plain; }
string PFCip::getKunci() { return this->kunci; }
string PFCip::getCipher() { return this->cipher; }
string PFCip::getFile() { return this->file; }

void PFCip::setPlain(string _plain) { this->plain = _plain; }
void PFCip::setKunci(string _kunci) { this->kunci = removeKarakterLain(_kunci); }
void PFCip::setCipher(string _cipher) { this->cipher = _cipher; }
void PFCip::setFile(string _file) { this->file = _file; }
```

6. Affine Cipher - AFC.cpp

```cpp
#include "AFC.h"
#include <iostream>
#include <string>

using namespace std;

AFC::AFC() {
    plain = "null";
    kunci_m = "null";
    kunci_b = "null";
    cipher = "null";
    file = "null";
}

void AFC::AFC_Enkripsi()
{
    string plain2 = getPlain();
    //Remove karakter non-alphabetic
    string plain = removeKarakterLain(plain2);
    //Menyelaraskan kunci dengan panjang Plain text
    int m = stoi(this->getKunciM());
    int b = stoi(this->getKunciB());
    string tampungan2 = "";
    for (int i = 0; i < (int)plain.size(); i++) {
        int p = charToInt(plain[i], true);
        int c = ((m * p) + b) % 26;
        char cnya = intToChar(c, true);
        tampungan2.push_back(cnya);
    }
    string temp = filterOutput(tampungan2, 5);
    setCipher(temp);
}

void AFC::AFC_Enkripsi_File()
{
    string cipher = getCipher();
    setFile(cipher);
}
```

```cpp
void AFC::AFC_Dekripsi() {
    int k, pengali;
    bool found = false;
    string cipher2 = getCipher();
    //Remove karakter non-alphabetic
    string cipher = removeKarakterLain(cipher2);
    //Menyelaraskan kunci dengan panjang Plain text
    int m = stoi(this->getKunciM());
    int b = stoi(this->getKunciB());
    //Menyelaraskan kunci dengan panjang Plain text
    pengali = 1;
    while (!found) {
        if ((((pengali * 26) + 1) % 26 == 1) && (((pengali * 26) + 1) % m == 0)) {
            k = ((pengali * 26) + 1) / m;
            found = true;
        }
        pengali++;
    }
    string tampungan2 = "";
    for (int i = 0; i < (int)cipher.size(); i++) {
        int c = charToInt(cipher[i], true);
        int p = ((c - b) * (k));
        if (p < 0) {
            int faktor = ((-p) / 26) + 1;
            p += faktor * 26;
        }
        p %= 26;
        char pnya = intToChar(p, true);
        tampungan2.push_back(pnya);
    }
    setPlain(tampungan2);
}

void AFC::AFC_Dekripsi_File() {
    string plain = getPlain();
    setFile(plain);
}
```



```cpp
string AFC::getPlain() { return this->plain; }
string AFC::getKunciB() { return this->kunci_b; }
string AFC::getKunciM() { return this->kunci_m; }
string AFC::getCipher() { return this->cipher; }
string AFC::getFile() { return this->file; }

void AFC::setPlain(string _plain) { this->plain = _plain; }
void AFC::setKunciB(string _kunci) { this->kunci_b = _kunci; }
void AFC::setKunciM(string _kunci) { this->kunci_m = _kunci; }
void AFC::setCipher(string _cipher) { this->cipher = _cipher; }
void AFC::setFile(string _file) { this->file = _file; }
```

BONUS

7. Hill Cipher - HC.cpp

```cpp
#include "HCip.h"
#include <iostream>

 using namespace std;

HCip::HCip() {
    plain = "null";
    kunci = "null";
    cipher = "null";
    file = "null";
}

vector<int> HCip::kaliMatEnkripsi(vector<vector<int>> kunci, vector<int> plain) {
    vector<int> hasil(kunci.size(), 0);
    for (unsigned int i = 0; i < kunci.size(); i++) {
        for (int u = 0; u < kunci.size(); u++) {
            hasil[i] += kunci[i][u] * plain[u];
        }
        hasil[i] %= 26;
    }
    return hasil;
}
```

```cpp
void HCip::HCip_Enkripsi()
{
    string _kunci = this->getKunci();
    vector<char> v, v2;
    vector<vector<int>> k3;
    k3 = generateKunci(_kunci);
    string plain2 = getPlain();
    //Remove karakter non-alphabetic
    string plain = removeKarakterLain(plain2);
    //Proses enkripsi
    string cipher;
    unsigned int k = 0;
    int faktor = floor(plain.size() / 3);
    int sisa = plain.size() % 3;
    for (int i = 0; i < sisa; i++) {
        plain.push_back('x');
    }
    vector<int> temp_plain;
    vector<int> result;
    while (k < plain.size()) {
        temp_plain.push_back(charToInt(plain[k], true));
        if ((k % 3 == 2) && (temp_plain.size()==3)) {
            result = kaliMatEnkripsi(k3, temp_plain);
            for (int index = 0; index < 3; index++) {
                cipher.push_back(intToChar(result[index],true));
            }
            temp_plain.clear();
            result.clear();
    }
    k++;
    }
    for (int i = 0; i < sisa; i++) {
        plain.pop_back();
    }
    string temp = filterOutput(cipher, 5);
    setCipher(temp);
}
```

```cpp
void HCip::HCip_Enkripsi_File()
{
    string cipher = getCipher();
    setFile(cipher);
}

void HCip::HCip_Dekripsi() {
    string _kunci = this->getKunci();
    vector<char> v, v2;
    vector<vector<int>> k3, k2, k1, i1, i2, i3;
    k3 = generateKunci(_kunci);
    i3 = generateInvers(k3);
    string cipher = getCipher();
    //Remove karakter non-alphabetic
    cipher = removeKarakterLain(cipher);
    //Proses dekripsi
    string plain;
    unsigned int k = 0;
    int sisa = cipher.size() % 3;
    vector<int> temp_cipher;
    vector<int> result;
    while (k < cipher.size()) {
        temp_cipher.push_back(charToInt(cipher[k], true));
        if ((k % 3 == 2) && (temp_cipher.size() == 3)) {
            result = kaliMatEnkripsi(i3, temp_cipher);
            for (int index = 0; index < 3; index++) {
                plain.push_back(intToChar(result[index], true));
            }
            temp_cipher.clear();
            result.clear();
        }
        k++;
    }
    for (int i = plain.size()-1; i > 0; i--) {
        if (plain[i] == 'x') {
            plain.pop_back();
        }
    }
    setPlain(plain);
}
```

```cpp
void HCip::HCip_Dekripsi_File() {
    string plain = getPlain();
    setFile(plain);
}

vector<vector<int>> HCip::generateKunci(string _kunci)
{
    return vector<vector<int>>{
        {17, 17, 5},
        {21, 18, 21},
        {2, 2, 19}
    };
}
```

```cpp
vector<vector<int>> HCip::generateInvers(vector<vector<int>>matriks) {
    return vector<vector<int>>{
        {4,9,15},
        {15,17,6},
        {24,0,17}
    };
}

string HCip::getPlain() { return this->plain; }
string HCip::getKunci() { return this->kunci; }
string HCip::getCipher() { return this->cipher; }
string HCip::getFile() { return this->file; }

void HCip::setPlain(string _plain) { this->plain = _plain; }
void HCip::setKunci(string _kunci) { this->kunci = removeKarakterLain(_kunci); }
void HCip::setCipher(string _cipher) { this->cipher = _cipher; }
void HCip::setFile(string _file) { this->file = _file; }
```

8. Enigma Cipher - ECip.cpp

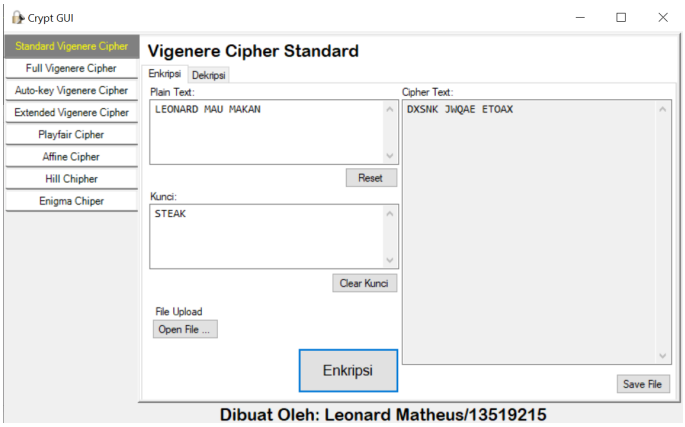
```
char ECip::ptoc(char karakter) {
    for (auto itr = m1.find(karakter); itr != m1.end(); itr++) {
        for (auto itr2 = m2.find(itr->second); itr2 != m2.end(); itr2++) {
            for (auto itr3 = m3.find(itr2->second); itr3 != m3.end(); itr3++) {
                for (auto itr4 = m4.find(itr3->second); itr4 != m4.end(); itr4++) {
                    for (auto itr5 = m5.find(itr4->second); itr5 != m5.end(); itr5++) {
                        for (auto itr6 = m6.find(itr5->second); itr6 != m6.end(); itr6++) {
                            for (auto itr7 = m7.find(itr6->second); itr7 != m7.end(); itr7++) {
                                return itr7->second;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

```
char ECip::ctop(char karakter) {
    for (auto& it7 : m7) {
        if (it7.second == karakter) {
            for (auto& it6 : m6) {
                if (it6.second == it7.first) {
                    for (auto& it5 : m5) {
                        if (it5.second == it6.first) {
                            for (auto& it4 : m4) {
                                if (it4.second == it5.first) {
                                    for (auto& it3 : m3) {
                                        if (it3.second == it4.first) {
                                            for (auto& it2 : m2) {
                                                if (it2.second == it3.first) {
                                                    for (auto& it1 : m1) {
                                                        if (it1.second == it2.first) {
                                                            return it1.first;
                                                        }
                                                    }
                                                }
                                            }
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

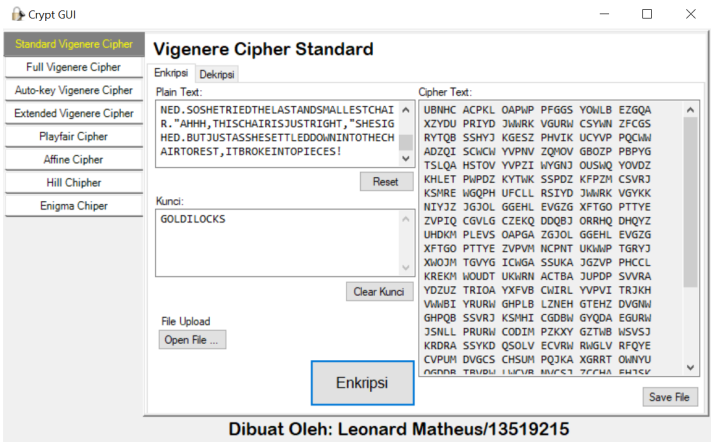
```
void ECip::ECip_Enkripsi()
{
        string plain2 = getPlain();
        //Remove karakter non-alphabetic
        string plain = removeKarakterLain(plain2);
        //Proses enkripsi
        string cipher;
        for (unsigned int i = 0; i < plain.size(); i++) {
                cipher.push_back(ptoc(plain[i]));
        }
```

```
        string temp = filterOutput(cipher, 5);
        setCipher(temp);
}

void ECip::ECip_Enkripsi_File()
{
        string cipher = getCipher();
        setFile(cipher);
}

void ECip::ECip_Dekripsi() {
        string _kunci = this->getKunci();
        string cipher = getCipher();
        //Remove karakter non-alphabetic
        cipher = removeKarakterLain(cipher);
        //Proses dekripsi
        string plain;
        for (unsigned int i = 0; i < cipher.size(); i++) {
                plain.push_back(ctop(cipher[i]));
        }
        setPlain(plain);
}

void ECip::ECip_Dekripsi_File() {
        string plain = getPlain();
        setFile(plain);
}

string ECip::getPlain() { return this->plain; }
string ECip::getKunci() { return this->kunci; }
string ECip::getCipher() { return this->cipher; }
string ECip::getFile() { return this->file; }

void ECip::setPlain(string _plain) { this->plain = _plain; }
void ECip::setKunci(string _kunci) { this->kunci = removeKarakterLain(_kunci); }
void ECip::setCipher(string _cipher) { this->cipher = _cipher; }
void ECip::setFile(string _file) { this->file = _file; }
```

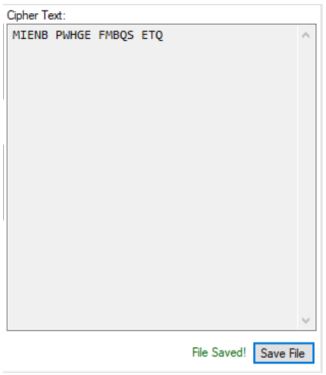
Tampilan Antarmuka Aplikasi



Tampilan Utama

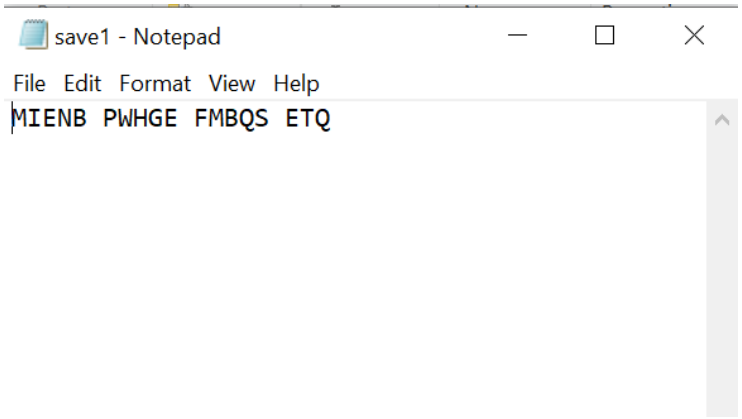


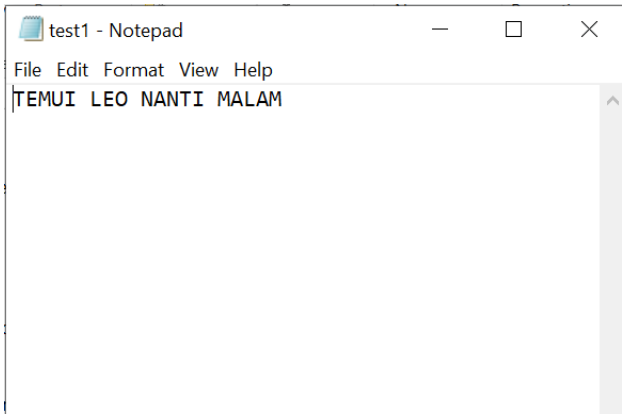
Handle Text Cipher yang banyak

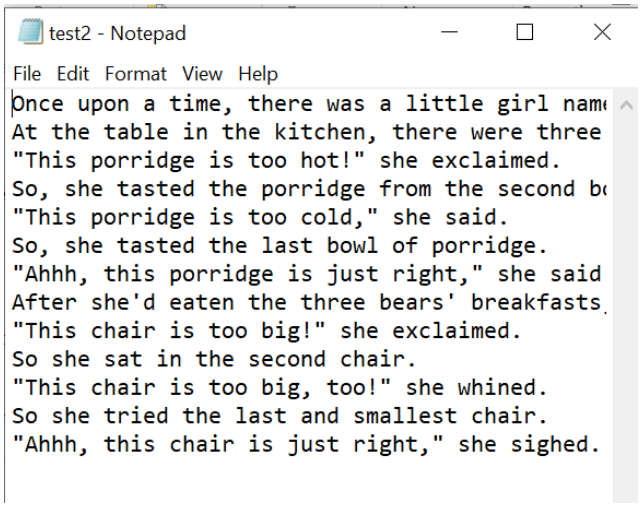


Fitur Open file dan save file

Contoh File







No.	Spek	Berhasil (✓)	Kurang Berhasil (✓)	Keterangan
1	Vigenere Cipher standard	✓		Text bisa enkripsi-dekripsi dan kembali ke asal
2	Full Vigenere Cipher	✓		Text bisa enkripsi-dekripsi dan kembali ke asal
3	Auto-Key Vigenere Cipher	✓		Text bisa enkripsi-dekripsi dan kembali ke asal
4	Extended Vigenere Cipher	✓		Text bisa enkripsi-dekripsi

				dan kembali ke asal
5	Playfair cipher	✓		Text bisa enkripsi-dekripsi dan kembali ke asal
6	Affine Cipher	✓		Text bisa enkripsi-dekripsi dan kembali ke asal
7	Hill Cipher	✓		Text bisa enkripsi-dekripsi dan kembali ke asal
8	Enigma cipher	✓		Text bisa enkripsi-dekripsi dan kembali ke asal

LETAK DRIVE UNTUK DIUJI COBA
https://github.com/leomatt547/Crypt_GUI