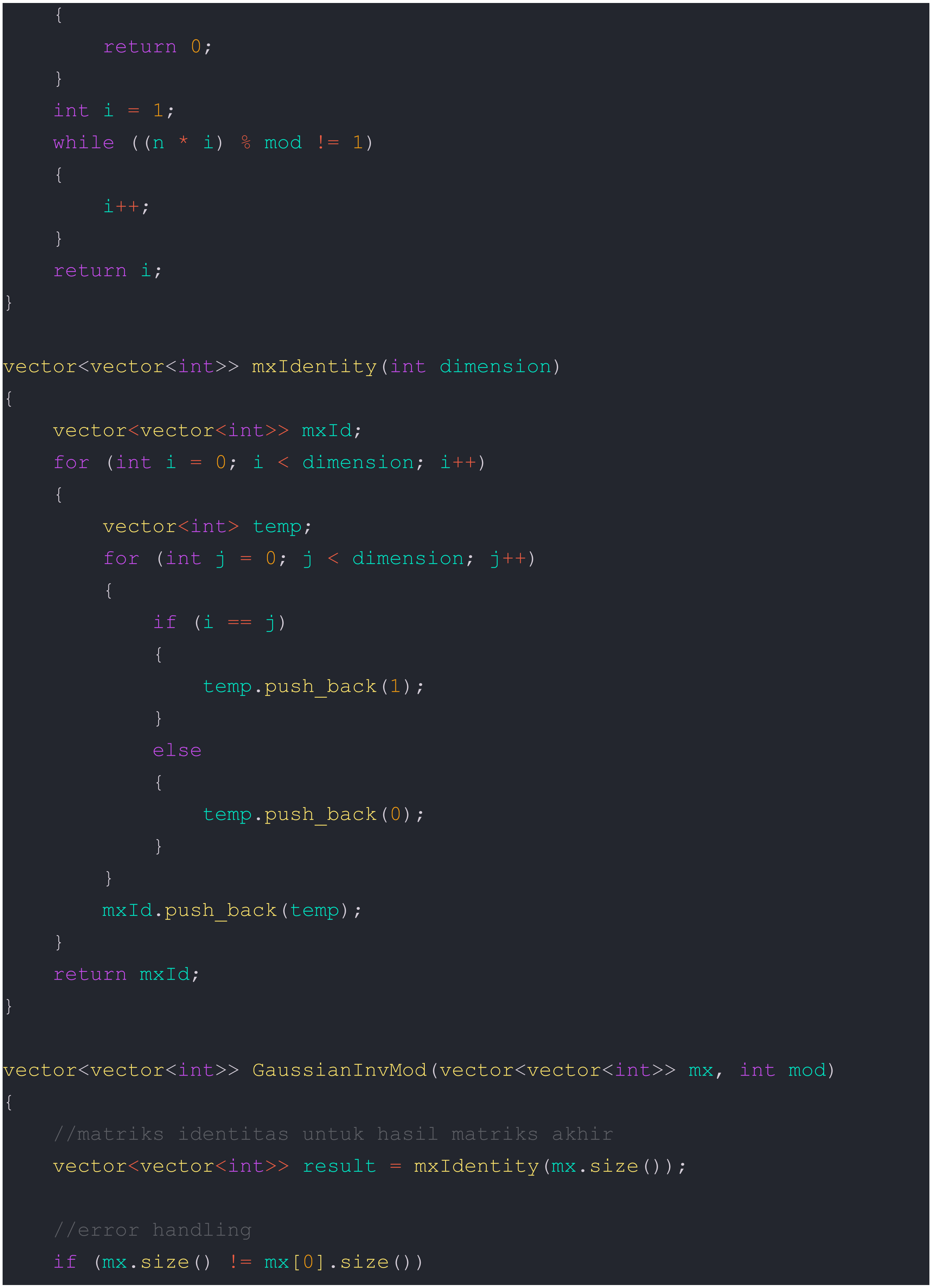
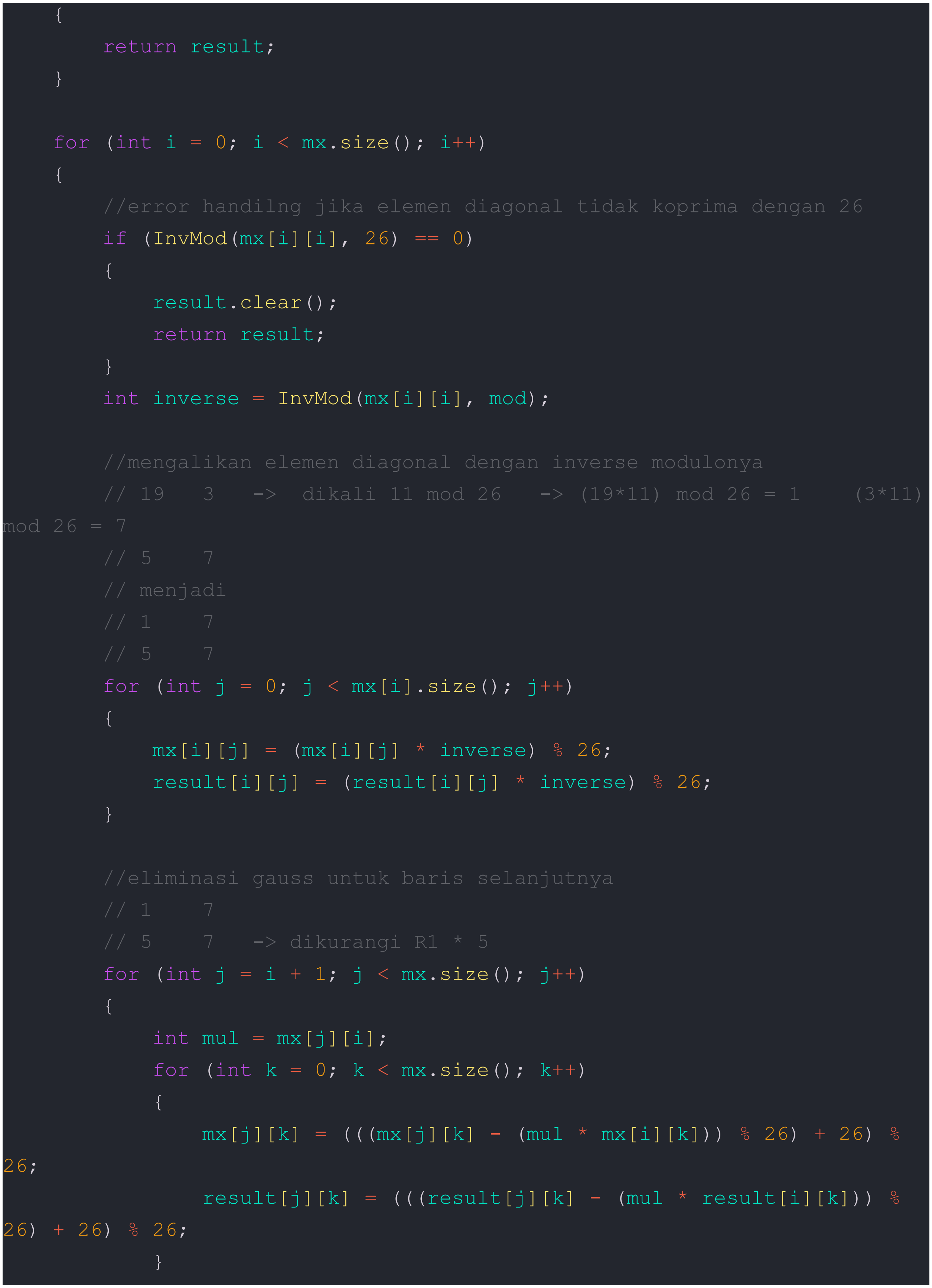
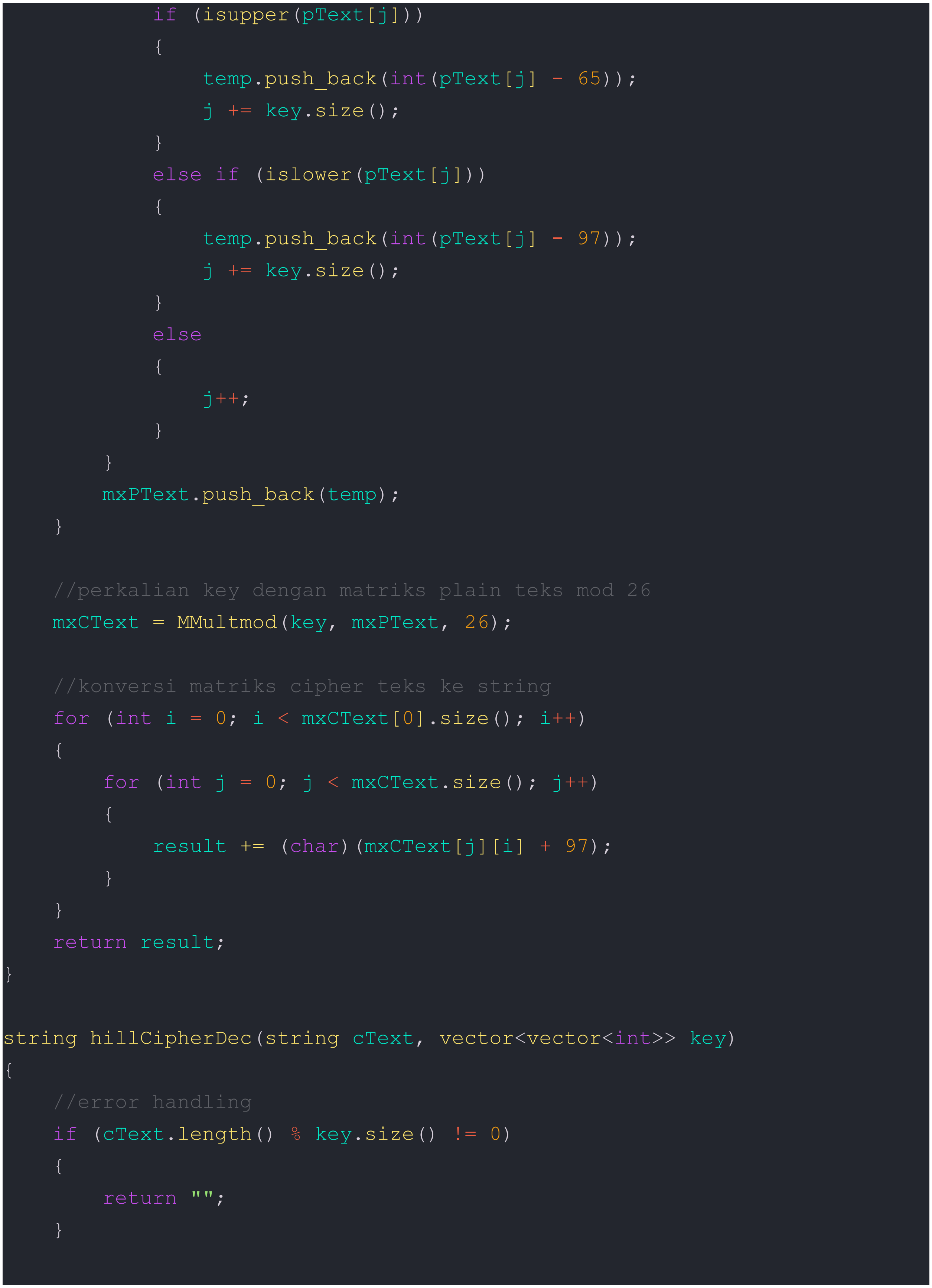
```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <math.h>

using namespace std;

vector<vector<int>> MMultmod(vector<vector<int>> a, vector<vector<int>> b, 
int mod)
{
    vector<vector<int>> result = {};
    if (a.empty() || b.empty())
    {
        return result;
    }
    if (a[0].size() != b.size())
    {
        return result;
    }
    for (int i = 0; i < a.size(); i++)
    {
        vector<int> temp;
        for (int j = 0; j < b[0].size(); j++)
        {
            int sum = 0;
            for (int k = 0; k < a[0].size(); k++)
            {
                sum += a[i][k] * b[k][j];
            }
            temp.push_back(((sum % 26) + 26) % 26);
        }
        result.push_back(temp);
    }
    return result;
}

int InvMod(int n, int mod)
{
    if (__gcd(n, mod) != 1)
```



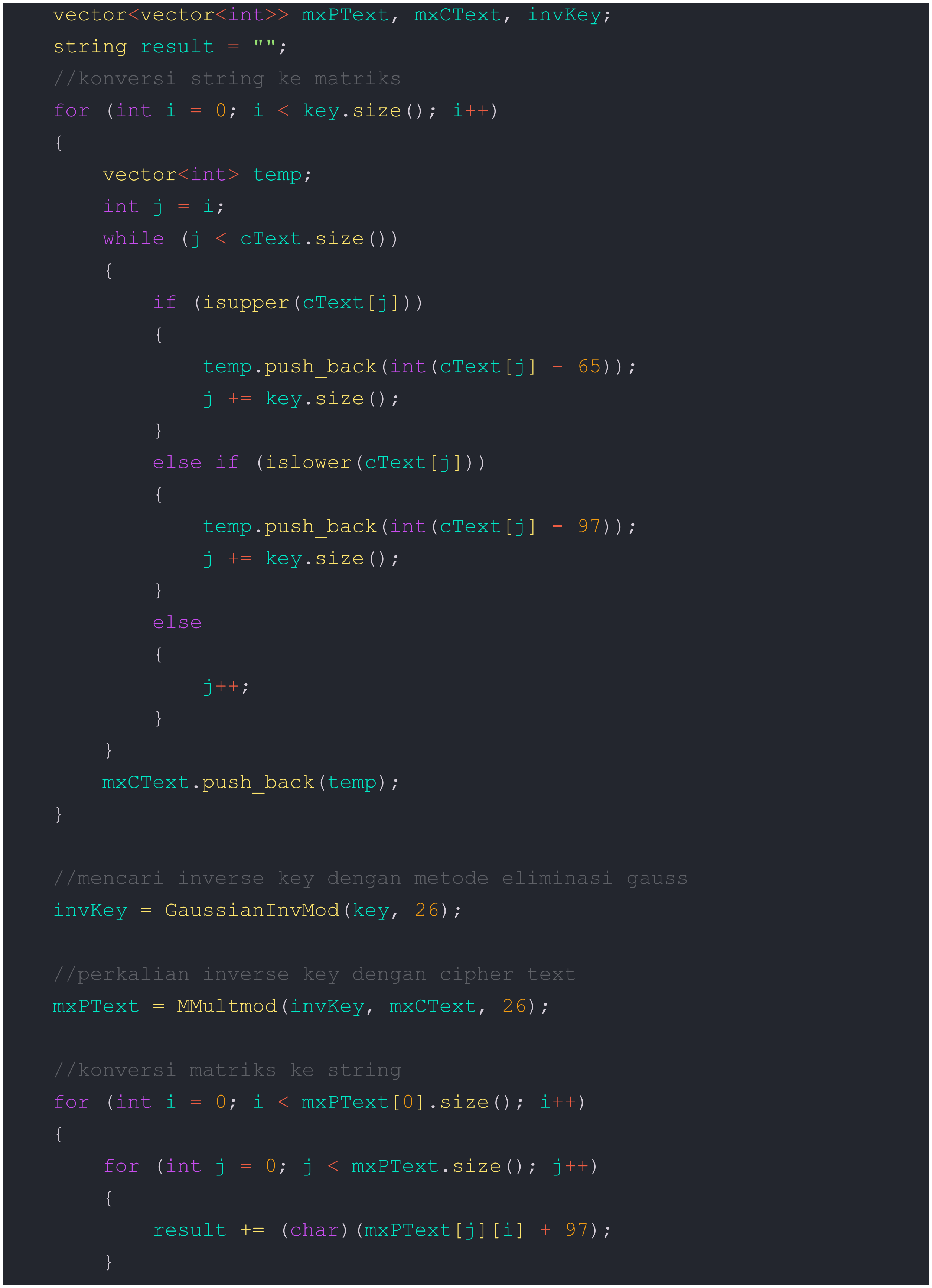
```
    {
        return 0;
    }
    int i = 1;
    while ((n * i) % mod != 1)
    {
        i++;
    }
    return i;
}

vector<vector<int>> mxIdentity(int dimension)
{
    vector<vector<int>> mxId;
    for (int i = 0; i < dimension; i++)
    {
        vector<int> temp;
        for (int j = 0; j < dimension; j++)
        {
            if (i == j)
            {
                temp.push_back(1);
            }
            else
            {
                temp.push_back(0);
            }
        }
        mxId.push_back(temp);
    }
    return mxId;
}

vector<vector<int>> GaussianInvMod(vector<vector<int>> mx, int mod)
{
    //matriks identitas untuk hasil matriks akhir
    vector<vector<int>> result = mxIdentity(mx.size());

    //error handling
    if (mx.size() != mx[0].size())
```



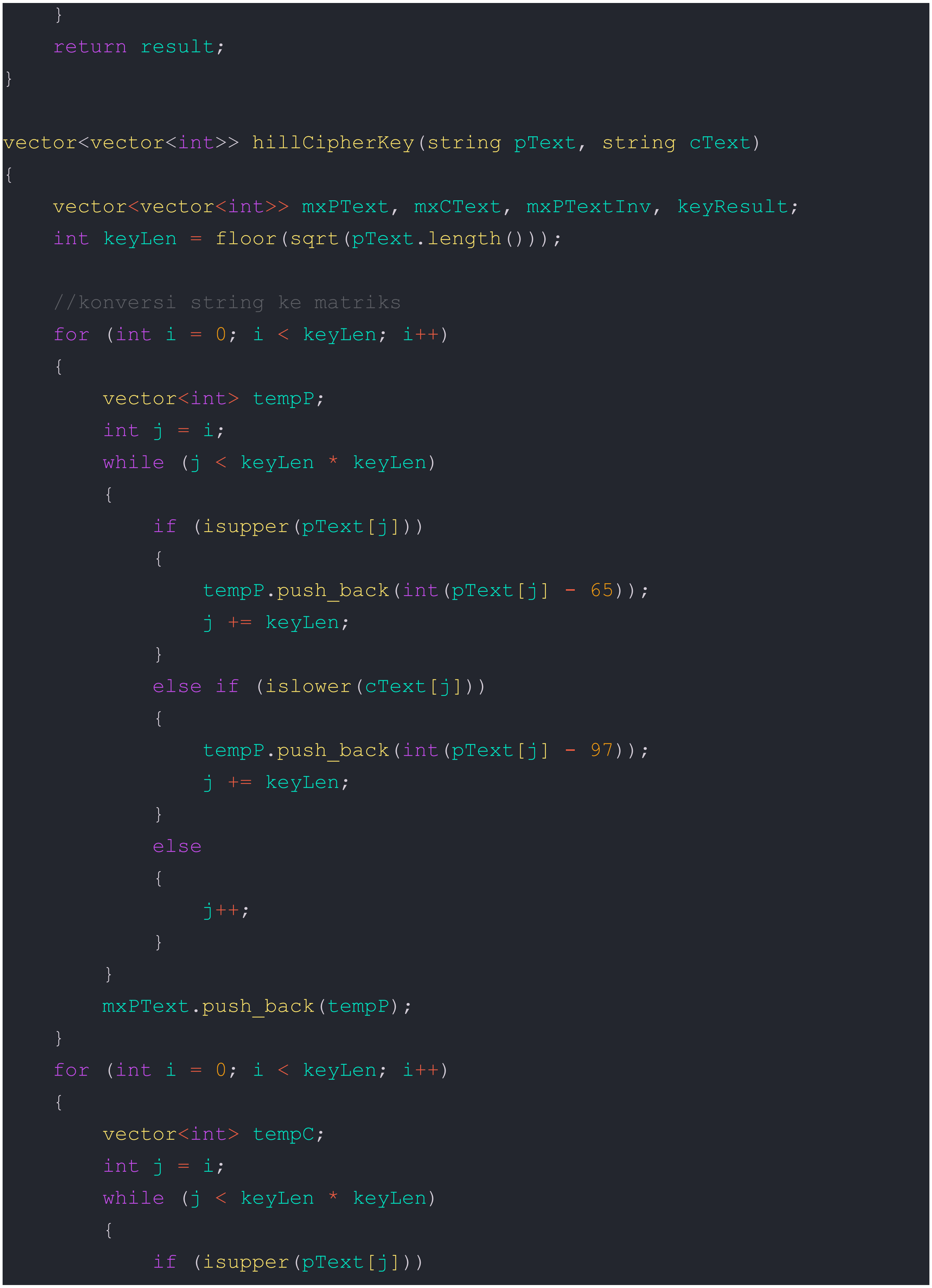
```
    {
        return result;
    }

    for (int i = 0; i < mx.size(); i++)
    {
        //error handilng jika elemen diagonal tidak koprima dengan 26
        if (InvMod(mx[i][i], 26) == 0)
        {
            result.clear();
            return result;
        }
        int inverse = InvMod(mx[i][i], mod);

        //mengalikan elemen diagonal dengan inverse modulonya
        // 19   3   ->  dikali 11 mod 26   -> (19*11) mod 26 = 1    (3*11)
mod 26 = 7
        // 5    7
        // menjadi
        // 1    7
        // 5    7
        for (int j = 0; j < mx[i].size(); j++)
        {
            mx[i][j] = (mx[i][j] * inverse) % 26;
            result[i][j] = (result[i][j] * inverse) % 26;
        }

        //eliminasi gauss untuk baris selanjutnya
        // 1    7
        // 5    7   -> dikurangi R1 * 5
        for (int j = i + 1; j < mx.size(); j++)
        {
            int mul = mx[j][i];
            for (int k = 0; k < mx.size(); k++)
            {
                mx[j][k] = (((mx[j][k] - (mul * mx[i][k])) % 26) + 26) %
26;
                result[j][k] = (((result[j][k] - (mul * result[i][k])) %
26) + 26) % 26;
            }
        }
    }
    //hasil akhir merupakan matriks segitiga atas dengan semua elemen diagonal bernilai 1

    //mengeliminasi matriks segitiga atas
    for (int i = mx.size() - 1; i >= 0; i--)
    {
        for (int j = i - 1; j >= 0; j--)
        {
            int mul = mx[j][i];
            for (int k = mx.size() - 1; k >= 0; k--)
            {
                mx[j][k] = (((mx[j][k] - (mul * mx[i][k])) % 26) + 26) % 26;
                result[j][k] = (((result[j][k] - (mul * result[i][k])) % 26) + 26) % 26;
            }
        }
    }
    return result;
}

string hillCipherEnc(string pText, vector<vector<int>> key)
{
    //error handling
    if (pText.length() % key.size() != 0)
    {
        return "";
    }

    vector<vector<int>> mxPText, mxCText;
    string result = "";
    //konversi string ke matriks
    for (int i = 0; i < key.size(); i++)
    {
        vector<int> temp;
        int j = i;
        while (j < pText.size())
        {
```



```
            if (isupper(pText[j]))
            {
                temp.push_back(int(pText[j] - 65));
                j += key.size();
            }
            else if (islower(pText[j]))
            {
                temp.push_back(int(pText[j] - 97));
                j += key.size();
            }
            else
            {
                j++;
            }
        }
        mxPText.push_back(temp);
    }

    //perkalian key dengan matriks plain teks mod 26
    mxCText = MMultmod(key, mxPText, 26);

    //konversi matriks cipher teks ke string
    for (int i = 0; i < mxCText[0].size(); i++)
    {
        for (int j = 0; j < mxCText.size(); j++)
        {
            result += (char)(mxCText[j][i] + 97);
        }
    }
    return result;
}

string hillCipherDec(string cText, vector<vector<int>> key)
{
    //error handling
    if (cText.length() % key.size() != 0)
    {
        return "";
    }
```

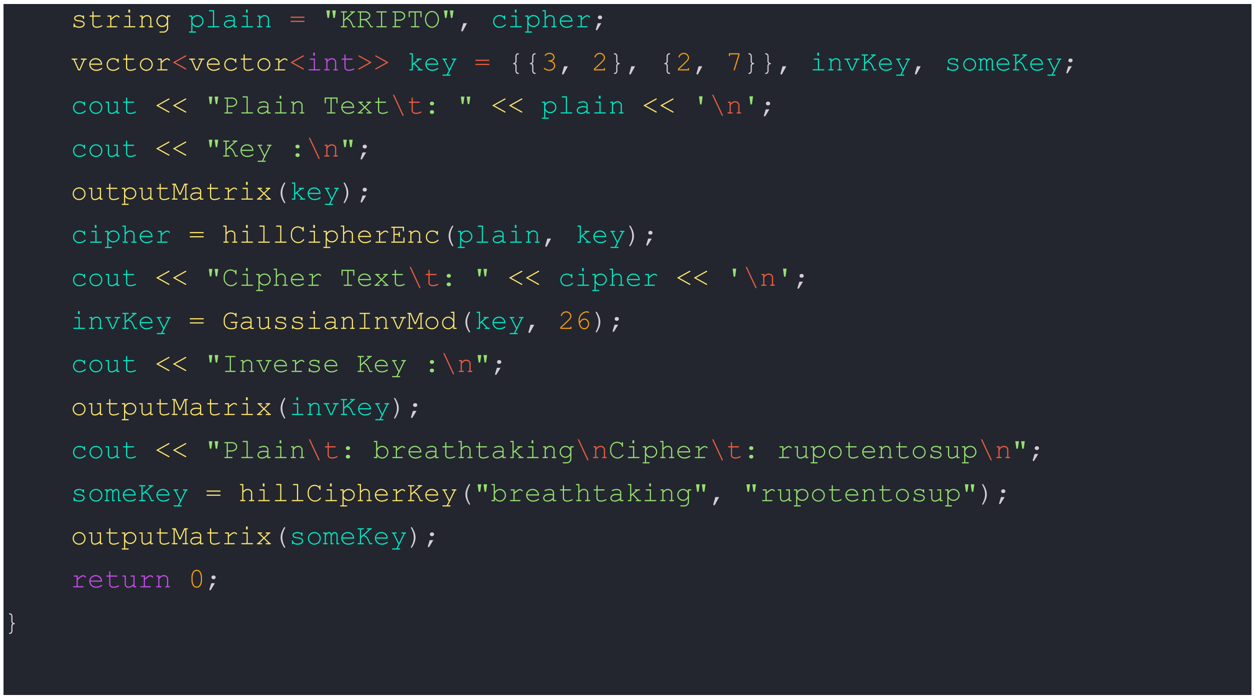


```
    vector<vector<int>> mxPText, mxCText, invKey;
    string result = "";
    //konversi string ke matriks
    for (int i = 0; i < key.size(); i++)
    {
        vector<int> temp;
        int j = i;
        while (j < cText.size())
        {
            if (isupper(cText[j]))
            {
                temp.push_back(int(cText[j] - 65));
                j += key.size();
            }
            else if (islower(cText[j]))
            {
                temp.push_back(int(cText[j] - 97));
                j += key.size();
            }
            else
            {
                j++;
            }
        }
        mxCText.push_back(temp);
    }

    //mencari inverse key dengan metode eliminasi gauss
    invKey = GaussianInvMod(key, 26);

    //perkalian inverse key dengan cipher text
    mxPText = MMultmod(invKey, mxCText, 26);

    //konversi matriks ke string
    for (int i = 0; i < mxPText[0].size(); i++)
    {
        for (int j = 0; j < mxPText.size(); j++)
        {
            result += (char)(mxPText[j][i] + 97);
        }
```



```
    }
    return result;
}

vector<vector<int>> hillCipherKey(string pText, string cText)
{
    vector<vector<int>> mxPText, mxCText, mxPTextInv, keyResult;
    int keyLen = floor(sqrt(pText.length()));

    //konversi string ke matriks
    for (int i = 0; i < keyLen; i++)
    {
        vector<int> tempP;
        int j = i;
        while (j < keyLen * keyLen)
        {
            if (isupper(pText[j]))
            {
                tempP.push_back(int(pText[j] - 65));
                j += keyLen;
            }
            else if (islower(cText[j]))
            {
                tempP.push_back(int(pText[j] - 97));
                j += keyLen;
            }
            else
            {
                j++;
            }
        }
        mxPText.push_back(tempP);
    }
    for (int i = 0; i < keyLen; i++)
    {
        vector<int> tempC;
        int j = i;
        while (j < keyLen * keyLen)
        {
            if (isupper(pText[j]))
```

```
                {
                    tempC.push_back(int(cText[j] - 65));
                    j += keyLen;
                }
                else if (islower(cText[j]))
                {
                    tempC.push_back(int(cText[j] - 97));
                    j += keyLen;
                }
                else
                {
                    j++;
                }
            }
            mxCText.push_back(tempC);
        }

    //mencari inverse matriks plain teks
    mxPTextInv = GaussianInvMod(mxPText, 26);

    //mencari key dengan mengalikan matriks cipher dengan matriks inverse plain mod 26
    keyResult = MMultmod(mxCText, mxPTextInv, 26);
    return keyResult;
}

void outputMatrix(vector<vector<int>> mx)
{
    for (int i = 0; i < mx[0].size(); i++)
    {
        for (int j = 0; j < mx.size(); j++)
        {
            cout << mx[i][j] << " ";
        }
        cout << '\n';
    }
}

int main(int argc, char const *argv[])
{
```



```
    string plain = "KRIPTO", cipher;
    vector<vector<int>> key = {{3, 2}, {2, 7}}, invKey, someKey;
    cout << "Plain Text\t: " << plain << '\n';
    cout << "Key :\n";
    outputMatrix(key);
    cipher = hillCipherEnc(plain, key);
    cout << "Cipher Text\t: " << cipher << '\n';
    invKey = GaussianInvMod(key, 26);
    cout << "Inverse Key :\n";
    outputMatrix(invKey);
    cout << "Plain\t: breathtaking\nCipher\t: rupotentosup\n";
    someKey = hillCipherKey("breathtaking", "rupotentosup");
    outputMatrix(someKey);
    return 0;
}
```