```cpp
#include <iostream>
#include <vector>
using std::cout;
using std::endl;
using std::vector;

class Matrix
{
private:
    vector<vector<int>> mat;
    unsigned rows;
    unsigned cols;

public:
    Matrix(unsigned _rows, unsigned _cols, const int &_initial)
    {
        mat.resize(_rows);
        for (unsigned i = 0; i < mat.size(); i++)
        {
            mat[i].resize(_cols);
        }
        rows = _rows;
        cols = _cols;
    }
    Matrix(const Matrix &rhs)
    {
        mat = rhs.mat;
        rows = rhs.get_rows();
        cols = rhs.get_cols();
    }
    Matrix &operator=(const Matrix &rhs)
    {
        if (&rhs == this)
            return *this;

        unsigned new_rows = rhs.get_rows();
        unsigned new_cols = rhs.get_cols();

        mat.resize(new_rows);
        for (unsigned i = 0; i < mat.size(); i++)
        {
            mat[i].resize(new_cols);
        }

        for (unsigned i = 0; i < new_rows; i++)
        {
```

```
            for (unsigned j = 0; j < new_cols; j++)
            {
                mat[i][j] = rhs(i, j);
            }
        }
        rows = new_rows;
        cols = new_cols;

        return *this;
    }

    Matrix operator+(const Matrix &rhs)
    {
        Matrix result(rows, cols, 0.0);
        for (unsigned i = 0; i < rows; i++)
        {
            for (unsigned j = 0; j < cols; j++)
            {
                result(i, j) = (this->mat[i][j] + rhs(i, j)) %
26;
            }
        }
        return result;
    }
    Matrix operator-(const Matrix &rhs)
    {
        Matrix result(rows, cols, 0.0);
        for (unsigned i = 0; i < rows; i++)
        {
            for (unsigned j = 0; j < cols; j++)
            {
                result(i, j) = (this->mat[i][j] - rhs(i, j)) %
26;
            }
        }
        return result;
    }
    Matrix operator*(const Matrix &rhs)
    {
        unsigned rows = rhs.get_rows();
        unsigned cols = rhs.get_cols();
        Matrix result(rows, cols, 0.0);
        for (unsigned i = 0; i < rows; i++)
        {
            for (unsigned j = 0; j < cols; j++)
            {
```

```
                for (unsigned k = 0; k < rows; k++)
                {
                    result(i, j) += (this->mat[i][k] * rhs(k,
j)) % 26;
                }
            }
        }
        for (unsigned i = 0; i < rows; i++)
        {
            for (unsigned j = 0; j < cols; j++)
            {
                result(i, j) = result(i, j) % 26;
            }
        }

        return result;
    }

    int &operator()(const unsigned &row, const unsigned &col)
    {
        return this->mat[row][col];
    }
    const int &operator()(const unsigned &row, const unsigned
&col) const
    {
        return this->mat[row][col];
    }

    unsigned get_rows() const
    {
        return this->rows;
    }
    unsigned get_cols() const
    {
        return this->cols;
    }
};

Matrix getInverse(const Matrix &m)
{
    Matrix result(m.get_rows(), m.get_cols(), 0.0);

    int det = m(0, 0) * (m(1, 1) * m(2, 2) - m(2, 1) * m(1, 2))
-
              m(0, 1) * (m(1, 0) * m(2, 2) - m(1, 2) * m(2, 0))
+
```

```
133                 m(0, 2) * (m(1, 0) * m(2, 1) - m(1, 1) * m(2,
    0));
134     det = det % 26;
135     int invdet = 0;
136     for (int i = 1; i <= 26; i++)
137     {
138         if (i * det % 26 == 1)
139         {
140             invdet = i;
141         }
142     }
143 
144     result(0, 0) = ((m(1, 1) * m(2, 2) - m(2, 1) * m(1, 2)) *
    invdet) % 26;
145     result(0, 1) = ((m(0, 2) * m(2, 1) - m(0, 1) * m(2, 2)) *
    invdet) % 26;
146     result(0, 2) = ((m(0, 1) * m(1, 2) - m(0, 2) * m(1, 1)) *
    invdet) % 26;
147     result(1, 0) = ((m(1, 2) * m(2, 0) - m(1, 0) * m(2, 2)) *
    invdet) % 26;
148     result(1, 1) = ((m(0, 0) * m(2, 2) - m(0, 2) * m(2, 0)) *
    invdet) % 26;
149     result(1, 2) = ((m(1, 0) * m(0, 2) - m(0, 0) * m(1, 2)) *
    invdet) % 26;
150     result(2, 0) = ((m(1, 0) * m(2, 1) - m(2, 0) * m(1, 1)) *
    invdet) % 26;
151     result(2, 1) = ((m(2, 0) * m(0, 1) - m(0, 0) * m(2, 1)) *
    invdet) % 26;
152     result(2, 2) = ((m(0, 0) * m(1, 1) - m(1, 0) * m(0, 1)) *
    invdet) % 26;
153 
154     return result;
155 }
156 
157 void modulo(Matrix &m)
158 {
159     for (int i = 0; i < 3; i++)
160     {
161         for (int j = 0; j < 3; j++)
162         {
163             if (m(i, j) < 0)
164             {
165                 m(i, j) = 26 + m(i, j);
166             }
167         }
168     }
```

```
169 }
170
171 void show(Matrix m, unsigned rows = 3, unsigned cols = 3)
172 {
173     for (unsigned i = 0; i < rows; i++)
174     {
175         for (unsigned j = 0; j < cols; j++)
176         {
177             cout << m(i, j) << " ";
178         }
179         cout << endl;
180     }
181 }
182
183 int main()
184 {
185     int x1[] = {0, 3, 8, 18, 15, 11, 0, 24, 4};
186     int x2[] = {3, 4, 16, 20, 0, 19, 8, 14, 13};
187     int y1[] = {3, 18, 17, 12, 18, 8, 14, 15, 11};
188     int y2[] = {23, 11, 9, 1, 25, 20, 11, 11, 12};
189     Matrix m1(3, 3, 0.0), m2(3, 3, 0.0), m3(3, 3, 0.0), m4(3,
    3, 0.0);
190     for (int i = 0; i < 3; i++)
191     {
192         for (int j = 0; j < 3; j++)
193         {
194             m1(i, j) = x1[3 * i + j];
195             m2(i, j) = x2[3 * i + j];
196             m3(i, j) = y1[3 * i + j];
197             m4(i, j) = y2[3 * i + j];
198         }
199     }
200     Matrix X = m2 - m1;
201     Matrix Y = m4 - m3;
202     cout << "X is \n";
203     show(X);
204     cout << "Y is \n";
205     show(Y);
206     Matrix invX = getInverse(X);
207     modulo(invX);
208     cout << "inverse of X is \n";
209     show(invX);
210     Matrix L = invX * Y;
211     modulo(L);
212     cout << "the key is \n";
213     show(L);
```

```
214       Matrix b = m3 - m1 * L;
215       modulo(b);
216       cout << "b is \n";
217       show(b, 1, 3);
218   }
```