matrix.cpp

```cpp
#include "matrix.h"

Matrix::Matrix () : m_row{3}, m_col{m_row} {
    _get_area();

    int count = 1;
    for (int i = 0; i < m_row; ++i) {
        for (int j = 0; j < m_col; ++j) {
            m_matrix[i][j] = count;
            ++count;
        }
    }
}

Matrix::Matrix (int row, int col) : m_row{row}, m_col{col} {
    _get_area();
    set_zero();
}

Matrix::Matrix (const Matrix& other) : m_row{other.m_row}, m_col{other.m_col} {
    _get_area();

    for (int i = 0; i < m_row; ++i) {
        for (int j = 0; j < m_col; ++j) {
            m_matrix[i][j] = other.m_matrix[i][j];
        }
    }
}

Matrix::~Matrix () noexcept {
    erase();
}

void Matrix::set_zero () {
    for (int i = 0; i < m_row; ++i) {
        for (int j = 0; j < m_col; ++j) {
            m_matrix[i][j] = 0;
        }
    }
}

void Matrix::set_random (int row, int col) {
    srand(time(0));
    if (row != m_row || col != m_col) {
        erase();
        m_row = row;
        m_col = col;
        _get_area();
    }
    _set_random();
}

void Matrix::set_by_input () {
    if (!empty())
        erase();
    std::cout << "Input the rows and cols: ";
    std::cin >> m_row >> m_col;
```

```cpp
    _get_area();

    for (int i = 0; i < m_row; ++i) {
        for (int j = 0; j < m_col; ++j) {
            std::cout << "Input the matrix[" << i << "][" << j << "] element: ";
            std::cin >> m_matrix[i][j];
        }
    }
}

int Matrix::get_row () const {
    return m_row;
}

int Matrix::get_col () const {
    return m_col;
}


void Matrix::display() const {

    int maxRowIndexDigits = 2;
    int maxColIndexDigits = std::to_string(m_col - 1).length();

    std::cout << "      ";
    for (int j = 0; j < m_col; ++j) {
        std::cout << std::setw(maxColIndexDigits + 1) << j + 1;
    }
    std::cout << '\n' << std::endl;
    for (int i = 0; i < m_row; ++i) {
        std::cout << std::setw(maxRowIndexDigits + 2) << i + 1 << " |";
        for (int j = 0; j < m_col; ++j) {
            std::cout << std::setw(maxColIndexDigits + 1) << m_matrix[i][j];
        }
        std::cout.put('\n');
    }
    std::cout << std::endl;
}

bool Matrix::empty () const {
    return !m_matrix;
}

bool Matrix::all_is_zero () const {
    for (int i = 0; i < m_row; ++i) {
        for (int j = 0; j < m_col; ++j) {
            if (m_matrix[i][j] != 0)
                return 0;
        }
    }
    return 1;
}

void Matrix::erase () noexcept {
    m_row = 0;
    m_col = 0;
    for (int i = 0; i < m_row; ++i) {
        delete m_matrix[i];
    }
    delete m_matrix;
```

```
118     m_matrix = nullptr;
119 }
120 
121 Matrix Matrix::operator+ (const Matrix& rhs) {
122     if (m_row != rhs.m_row || m_col != rhs.m_col)
123         throw std::invalid_argument("The cols and rows aren't equal");
124     Matrix other(*this);
125     for (int i = 0; i < m_row; ++i) {
126         for (int j = 0; j < m_col; ++j) {
127             other.m_matrix[i][j] += rhs.m_matrix[i][j];
128         }
129     }
130     return other;
131 }
132 
133 const Matrix& Matrix::operator= (const Matrix& rhs) {
134     if (this == &rhs)
135         return *this;
136     if (rhs.m_row != m_row || rhs.m_col != m_col) {
137         erase();
138         m_row = rhs.m_row;
139         m_col = rhs.m_col;
140         _get_area();
141     }
142     for (int i = 0; i < m_row; ++i) {
143         for (int j = 0; j < m_col; ++j) {
144             m_matrix[i][j] = rhs.m_matrix[i][j];
145         }
146     }
147     return *this;
148 }
149 
150 Matrix Matrix::operator* (const Matrix& rhs) {
151     if (m_col != rhs.m_row) // ստուգում ենք առաջին մատրիցի սյուների և երկրորդ
մատրիցի տողերի համապատասխանությունը
152         throw std::invalid_argument("the cols must be equal to rows");
153 
154     Matrix matrix;
155     matrix.set_random(m_row, rhs.m_col);
156     matrix.set_zero();
157     for (int i = 0; i < m_row; ++i) {
158         for (int j = 0; j < rhs.m_col; ++j) {
159             for (int k = 0; k < m_col; ++k) { // օգտագործում ենք մատրիցների
արտադրյալի բանաձևը
160                 matrix.m_matrix[i][j] += (m_matrix[i][k] * rhs.m_matrix[k][j]);
161             }
162         }
163     }
164     return matrix;
165 }
166 
167 void Matrix::operator+= (const Matrix& rhs) {
168     for (int i = 0; i < m_row; ++i) {
169         for (int j = 0; j < m_col; ++j) {
170             m_matrix[i][j] += rhs.m_matrix[i][j];
171         }
172     }
173 }
174 
175 const int& Matrix::operator() (int i, int j) const {
```

```cpp
    return m_matrix[i][j];
}

int& Matrix::operator() (int i, int j) {
    return const_cast<int&>(static_cast<const Matrix&>(*this)(i, j));
}

void Matrix::_set_random () {
    for (int i = 0; i < m_row; ++i) {
        for (int j = 0; j < m_col; ++j) {
            m_matrix[i][j] = rand () % 6;
        }
    }
}

void Matrix::_get_area () {
    m_matrix = new int*[m_row];
    for (int i = 0; i < m_row; ++i) {
        m_matrix[i] = new int[m_col];
    }
}


```