**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСТИЕТ**

Лабораторная работа №4 по дисциплине «Программирование»

**Наследование**

Группа: **АВТ-342**

Студенты: **Бондаренко А.В.,Фадеев В.А.**

НОВОСИБИРСК 2024

**1.Постановка задачи**

**Вариант 4.**

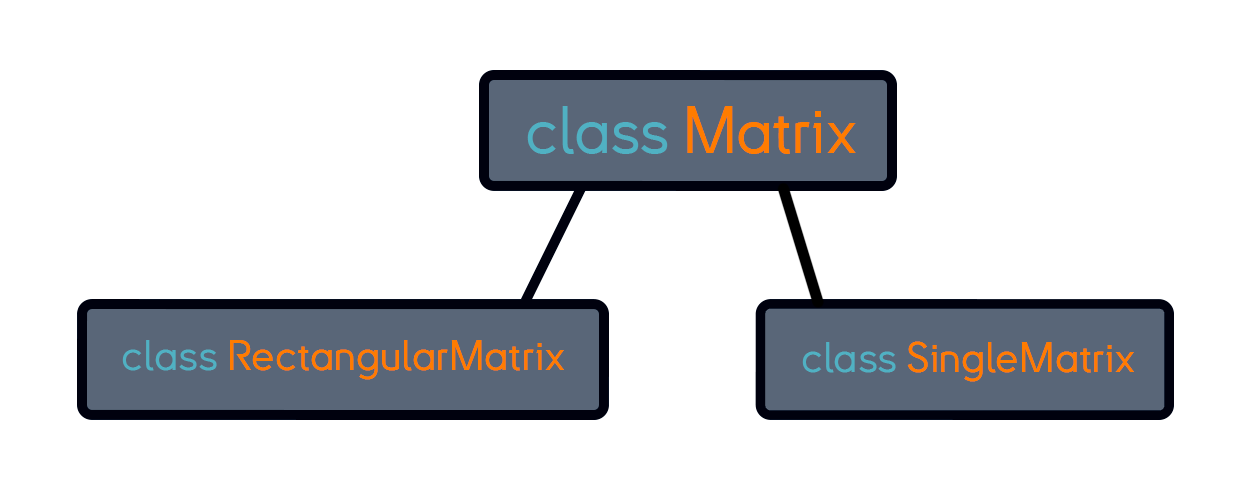
Для классов предыдущей лабораторной работы реализовать иерархию, перегрузив отдельные методы и добавляя члены-данные и методы по заданию и/или усмотрению студента. В иерархию должно входить 2-3 производных класса.

Изменить демонстрационную программу так, чтобы она демонстрировала создание, копирование объектов родственных типов.

Класс для pаботы с квадратными матрицами. Создать наследников класса:

* класс - единичная матрица;
* класс для pаботы с прямоугольными матрицами.

**2. Отображение иерархии в виде графа**

****

### 3. Определение пользовательских классов с комментариями: Класс Matrix

* **Описание**: Этот класс представляет матрицу произвольного размера.
* **Члены класса**:
  + int rows: количество строк в матрице.
  + int cols: количество столбцов в матрице.
  + double\*\* data: двумерный массив для хранения элементов матрицы.
* **Методы**:
  + Matrix(int r, int c): Конструктор, который инициализирует матрицу заданного размера и заполняет её нулями.
  + ~Matrix(): Деструктор, освобождающий память.
  + int getRows() const: Возвращает количество строк матрицы.
  + int getCols() const: Возвращает количество столбцов матрицы.
  + void set(int r, int c, double value): Устанавливает значение элемента матрицы по заданным индексам.
  + double get(int r, int c) const: Возвращает значение элемента матрицы по заданным индексам.
  + void display() const: Отображает матрицу на экране.
  + Matrix operator+(const Matrix& other) const: Перегруженный оператор сложения, который складывает две матрицы.
  + Matrix operator-(const Matrix& other) const: Перегруженный оператор вычитания, который вычитает одну матрицу из другой.
  + Matrix operator\*(const Matrix& other) const: Перегруженный оператор умножения, который умножает две матрицы.
  + Matrix transpose() const: Возвращает транспонированную матрицу.
  + void saveToFile(std::ofstream& out) const: Сохраняет матрицу в текстовый файл.
  + void loadFromFile(std::ifstream& in): Загружает матрицу из текстового файла.
  + void saveBinary(std::ofstream& out) const: Сохраняет матрицу в бинарный файл.
  + void loadBinary(std::ifstream& in): Загружает матрицу из бинарного файла.

**Класс SingleMatrix (наследуется от Matrix)**

* **Описание**: Этот класс представляет матрицу, состоящую из единственного элемента.
* **Методы**:
  + SingleMatrix(double value): Конструктор, который инициализирует матрицу с одним элементом.
  + Matrix operator+(const Matrix& other) const: Перегруженный оператор сложения, который складывает единственную матрицу с другой матрицей.
  + Matrix operator-(const Matrix& other) const: Перегруженный оператор вычитания.
  + Matrix operator\*(const Matrix& other) const: Перегруженный оператор умножения.

**Класс RectangularMatrix (наследуется от Matrix)**

* **Описание**: Этот класс представляет прямоугольную матрицу, которая может иметь разное количество строк и столбцов.
* **Методы**:
  + RectangularMatrix(int r, int c): Конструктор, который инициализирует прямоугольную матрицу заданного размера.
  + Matrix operator+(const Matrix& other) const: Перегруженный оператор сложения для прямоугольной матрицы.
  + Matrix operator-(const Matrix& other) const: Перегруженный оператор вычитания.
  + Matrix operator\*(const Matrix& other) const: Перегруженный оператор умножения.

Эта структура классов позволяет создавать и работать с матрицами произвольного размера, а также с матрицами, состоящими из единственного элемента и прямоугольными матрицами.

**Вывод**В ходе выполнения лабораторной работы была разработана программа для работы с матрицами, реализующая иерархию классов для квадратных, единичных и прямоугольных матриц. Программа включает в себя операции сложения, вычитания, умножения, транспонирования, а также функции для сохранения и загрузки матриц в текстовые и бинарные файлы.

Создан класс Matrix, который служит базовым для производных классов SingleMatrix и RectangularMatrix. Этот базовый класс предоставляет методы для выполнения основных математических операций и управление данными матрицы. В производных классах были переопределены методы, обеспечивающие специфичное поведение для единичных и прямоугольных матриц, а также была добавлена возможность удобной работы с объектами этих классов через перегрузку операторов.  
**Текст программы с комментариями**

**Main.cpp**

#include "matrix.h"  
#include <iostream>  
#include <fstream>  
using namespace std;  
  
int main() {  
 Matrix result;  
 bool is\_result\_stored = false;  
 while (true) {  
 int MatrixType;  
 cout << "Choose type of Matrix:" << endl;  
 cout << "1. Square" << endl;  
 cout << "2. Single" << endl;  
 cout << "3. Rectangular" << endl;  
 cout << "0. Exit" << endl;  
 cin >> MatrixType;  
 if (MatrixType == 1) {  
 while (true) {  
 int choice;  
 cout << "Choose option:" << endl;  
 cout << "1. Matrix addition" << endl;  
 cout << "2. Matrix subtraction" << endl;  
 cout << "3. Matrix multiplication" << endl;  
 cout << "4. Transposition" << endl;  
 cout << "5. Save to file" << endl;  
 cout << "6. Load from file" << endl;  
 cout << "7. Save to bin file" << endl;  
 cout << "8. Load from bin file" << endl;  
 cout << "0. Exit" << endl;  
 cin >> choice;  
 if (choice == 0) {  
 break;  
 }  
 int rows, cols;  
 switch (choice) {  
 case 1: { // Сложение матриц  
 cout << "Size of square matrices: ";  
 while (1) {  
 cin >> rows >> cols;  
 if (rows == cols) {  
 break;  
 } else {  
 cout << "Rows and cols are not equal. Enter it again." << endl;  
 }  
 }  
 Matrix matrix1(rows, cols), matrix2(rows, cols);  
 cin >> matrix1 >> matrix2;  
 result = matrix1 + matrix2;  
 cout << "Result of addition:" << endl << result;  
 is\_result\_stored = true; // Результат сохранен  
 break;  
 }  
 case 2: { // Вычитание матриц  
 cout << "Size of square matrices: ";  
 while (1) {  
 cin >> rows >> cols;  
 if (rows == cols) {  
 break;  
 } else {  
 cout << "Rows and cols are not equal. Enter it again." << endl;  
 }  
 }  
 Matrix matrix1(rows, cols), matrix2(rows, cols);  
 cin >> matrix1 >> matrix2;  
 result = matrix1 - matrix2;  
 cout << "Result of subtraction:" << endl << result;  
 is\_result\_stored = true;  
 break;  
 }  
 case 3: { // Умножение матриц  
 int rows1, cols1, rows2, cols2;  
 cout << "Size of first matrix (rows columns): ";  
 while (1) {  
 cin >> rows1 >> cols1;  
 if (rows1 == cols1) {  
 break;  
 } else {  
 cout << "Rows and cols are not equal. Enter it again." << endl;  
 }  
 }  
 Matrix matrix1(rows1, cols1);  
 cout << "Size of second matrix (rows columns): ";  
 while (1) {  
 cin >> rows2 >> cols2;  
 if (rows2 == cols2) {  
 break;  
 } else {  
 cout << "Rows and cols are not equal. Enter it again." << endl;  
 }  
 }  
 Matrix matrix2(rows2, cols2);  
 if (cols1 != rows2) {  
 cout << "Matrices can't be multiplied" << endl;  
 break;  
 }  
 cin >> matrix1 >> matrix2;  
 result = matrix1.multiply(matrix2);  
 cout << "Result of multiplication:" << endl << result;  
 is\_result\_stored = true;  
 break;  
 }  
 case 4: { // Транспонирование матрицы  
 cout << "Size of matrix: ";  
 while (1) {  
 cin >> rows >> cols;  
 if (rows == cols) {  
 break;  
 } else {  
 cout << "Rows and cols are not equal. Enter it again." << endl;  
 }  
 }  
 Matrix matrix1(rows, cols), matrix2(rows, cols);  
 cin >> matrix1;  
 result = matrix1.transpose();  
 cout << "Result of transposition:" << endl << result;  
 is\_result\_stored = true;  
 break;  
 }  
 case 5: { // Сохранение результата в файл  
 if (is\_result\_stored) {  
 ofstream ofs("matrix.txt");  
 ofs << result;  
 ofs.close();  
 cout << "Result saved to file." << endl;  
 } else {  
 cout << "No result to save!" << endl;  
 }  
 break;  
 }  
 case 6: { // Загрузка матрицы из файла  
 ifstream ifs("matrix.txt");  
 if (ifs.is\_open()) {  
 ifs >> result;  
 cout << "Matrix loaded from file: " << "\n" << result << endl;  
 is\_result\_stored = true;  
 ifs.close();  
 } else {  
 cout << "Unable to open file" << endl;  
 }  
 break;  
 }  
 case 7: { // Сохранение результата в бинарный файл  
 if (is\_result\_stored) {  
 result.saveToBinary("matrix.bin");  
 } else {  
 cout << "No result to save!" << endl;  
 }  
 break;  
 }  
 case 8: { // Загрузка матрицы из бинарного файла  
 result.loadFromBinary("matrix.bin");  
 is\_result\_stored = true;  
 result.print();  
 break;  
 }  
 default:  
 cout << "Invalid option, try again." << endl;  
 break;  
 }  
 }  
 }  
 if (MatrixType == 2) {  
 while(true) {  
 int choice;  
 cout << "Choose option:" << endl;  
 cout << "1. Matrix addition" << endl;  
 cout << "2. Matrix subtraction" << endl;  
 cout << "3. Matrix multiplication" << endl;  
 cout << "4. Transposition" << endl;  
 cout << "5. Save to file" << endl;  
 cout << "6. Load from file" << endl;  
 cout << "7. Save to bin file" << endl;  
 cout << "8. Load from bin file" << endl;  
 cout << "0. Exit" << endl;  
 cin >> choice;  
 if (choice == 0) {  
 break;  
 }  
 int rows, cols;  
 switch (choice) {  
 case 1: { // Сложение матриц  
 rows = 1;  
 cols = 1;  
 SingleMatrix matrix1(rows, cols), matrix2(rows, cols);  
 cin >> matrix1 >> matrix2;  
 result = matrix1 + matrix2;  
 cout << "Result of addition:" << endl << result;  
 is\_result\_stored = true; // Результат сохранен  
 break;  
 }  
 case 2: { // Вычитание матриц  
 rows = 1;  
 cols = 1;  
 SingleMatrix matrix1(rows, cols), matrix2(rows, cols);  
 cin >> matrix1 >> matrix2;  
 result = matrix1 - matrix2;  
 cout << "Result of subtraction:" << endl << result;  
 is\_result\_stored = true;  
 break;  
 }  
 case 3: { // Умножение матриц  
 int rows1, cols1, rows2, cols2;  
 rows1 = 1;  
 cols1 = 1;  
 SingleMatrix matrix1(rows1, cols1);  
 cout << "Size of second matrix (rows columns): ";  
 rows2 = 1;  
 cols2 = 1;  
 SingleMatrix matrix2(rows2, cols2);  
 if (cols1 != rows2) {  
 cout << "Matrices can't be multiplied" << endl;  
 break;  
 }  
 cin >> matrix1 >> matrix2;  
 result = matrix1.multiply(matrix2);  
 cout << "Result of multiplication:" << endl << result;  
 is\_result\_stored = true;  
 break;  
 }  
 case 4: { // Транспонирование матрицы  
 rows = 1;  
 cols = 1;  
 SingleMatrix matrix1(rows, cols), matrix2(rows, cols);  
 cin >> matrix1;  
 result = matrix1.transpose();  
 cout << "Result of transposition:" << endl << result;  
 is\_result\_stored = true;  
 break;  
 }  
 case 5: { // Сохранение результата в файл  
 if (is\_result\_stored) {  
 ofstream ofs("matrix.txt");  
 ofs << result;  
 ofs.close();  
 cout << "Result saved to file." << endl;  
 } else {  
 cout << "No result to save!" << endl;  
 }  
 break;  
 }  
 case 6: { // Загрузка матрицы из файла  
 ifstream ifs("matrix.txt");  
 if (ifs.is\_open()) {  
 ifs >> result;  
 cout << "Matrix loaded from file: " << "\n" << result << endl;  
 is\_result\_stored = true;  
 ifs.close();  
 } else {  
 cout << "Unable to open file" << endl;  
 }  
 break;  
 }  
 case 7: { // Сохранение результата в бинарный файл  
 if (is\_result\_stored) {  
 result.saveToBinary("matrix.bin");  
 } else {  
 cout << "No result to save!" << endl;  
 }  
 break;  
 }  
 case 8: { // Загрузка матрицы из бинарного файла  
 result.loadFromBinary("matrix.bin");  
 is\_result\_stored = true;  
 result.print();  
 break;  
 }  
 default:  
 cout << "Invalid option, try again." << endl;  
 break;  
 }  
 }  
  
 }  
 if (MatrixType == 3) {  
 while(true) {  
 int choice;  
 cout << "Choose option:" << endl;  
 cout << "1. Matrix addition" << endl;  
 cout << "2. Matrix subtraction" << endl;  
 cout << "3. Matrix multiplication" << endl;  
 cout << "4. Transposition" << endl;  
 cout << "5. Save to file" << endl;  
 cout << "6. Load from file" << endl;  
 cout << "7. Save to bin file" << endl;  
 cout << "8. Load from bin file" << endl;  
 cout << "0. Exit" << endl;  
 cin >> choice;  
 if (choice == 0) {  
 break;  
 }  
 int rows, cols;  
 switch (choice) {  
 case 1: { // Сложение матриц  
 cout << "Size of matrices: ";  
 while (1) {  
 cin >> rows >> cols;  
 if (rows != cols) {  
 break;  
 } else {  
 cout << "Rows and cols are equal. Enter it again." << endl;  
 }  
 }  
 RectangularMatrix matrix1(rows, cols), matrix2(rows, cols);  
 cin >> matrix1 >> matrix2;  
 result = matrix1 + matrix2;  
 cout << "Result of addition:" << endl << result;  
 is\_result\_stored = true;  
 break;  
 }  
 case 2: { // Вычитание матриц  
 cout << "Size of matrices: ";  
 while (1) {  
 cin >> rows >> cols;  
 if (rows != cols) {  
 break;  
 } else {  
 cout << "Rows and cols are equal. Enter it again." << endl;  
 }  
 }  
 RectangularMatrix matrix1(rows, cols), matrix2(rows, cols);  
 cin >> matrix1 >> matrix2;  
 result = matrix1 - matrix2;  
 cout << "Result of subtraction:" << endl << result;  
 is\_result\_stored = true;  
 break;  
 }  
 case 3: { // Умножение матриц  
 int rows1, cols1, rows2, cols2;  
 cout << "Size of first matrix: ";  
 while (1) {  
 cin >> rows1 >> cols1;  
 if (rows1 != cols1) {  
 break;  
 } else {  
 cout << "Rows and cols are equal. Enter it again." << endl;  
 }  
 }  
 RectangularMatrix matrix1(rows1, cols1);  
 cout << "Size of second matrix: ";  
 while (1) {  
 cin >> rows2 >> cols2;  
 if (rows2 != cols2) {  
 break;  
 } else {  
 cout << "Rows and cols are equal. Enter it again." << endl;  
 }  
 }  
 RectangularMatrix matrix2(rows2, cols2);  
 if (cols1 != rows2) {  
 cout << "Matrices can't be multiplied" << endl;  
 break;  
 }  
 cin >> matrix1 >> matrix2;  
 result = matrix1.multiply(matrix2);  
 cout << "Result of multiplication:" << endl << result;  
 is\_result\_stored = true;  
 break;  
 }  
 case 4: { // Транспонирование матрицы  
 cout << "Size of matrix: ";  
 while (1) {  
 cin >> rows >> cols;  
 if (rows != cols) {  
 break;  
 } else {  
 cout << "Rows and cols are equal. Enter it again." << endl;  
 }  
 }  
 RectangularMatrix matrix1(rows, cols), matrix2(rows, cols);  
 cin >> matrix1;  
 result = matrix1.transpose();  
 cout << "Result of transposition:" << endl << result;  
 is\_result\_stored = true;  
 break;  
 }  
 case 5: { // Сохранение результата в файл  
 if (is\_result\_stored) {  
 ofstream ofs("matrix.txt");  
 ofs << result;  
 ofs.close();  
 cout << "Result saved to file." << endl;  
 } else {  
 cout << "No result to save!" << endl;  
 }  
 break;  
 }  
 case 6: { // Загрузка матрицы из файла  
 ifstream ifs("matrix.txt");  
 if (ifs.is\_open()) {  
 ifs >> result;  
 cout << "Matrix loaded from file: " << "\n" << result << endl;  
 is\_result\_stored = true;  
 ifs.close();  
 } else {  
 cout << "Unable to open file" << endl;  
 }  
 break;  
 }  
 case 7: { // Сохранение результата в бинарный файл  
 if (is\_result\_stored) {  
 result.saveToBinary("matrix.bin");  
 } else {  
 cout << "No result to save!" << endl;  
 }  
 break;  
 }  
 case 8: { // Загрузка матрицы из бинарного файла  
 result.loadFromBinary("matrix.bin");  
 is\_result\_stored = true;  
 result.print();  
 break;  
 }  
 default:  
 cout << "Invalid option, try again." << endl;  
 break;  
 }  
 }  
 }  
 if (MatrixType == 0) {  
 break;  
 }  
 }  
  
 return 0;  
}

**Matrix.h**

#ifndef MATRIX\_H  
#define MATRIX\_H  
  
#include <iostream>  
#include <fstream>  
using namespace std;  
  
// Базовый класс для матриц  
class Matrix {  
public:  
 Matrix(); // Конструктор по умолчанию  
 Matrix(int rows, int cols); // Конструктор с заданием размеров  
 Matrix(const Matrix&); // Конструктор копирования  
 ~Matrix(); // Деструктор  
 Matrix multiply(const Matrix&) const; // Умножение матриц  
 Matrix transpose() const; // Транспонирование матрицы  
 Matrix operator=(const Matrix&); // Оператор присваивания  
 Matrix operator-(const Matrix&) const; // Оператор вычитания  
 int operator()() const; // Оператор для получения размеров  
 int& operator()(int row, int column); // Оператор доступа к элементам  
 friend Matrix operator+(const Matrix&, const Matrix&); // Дружественный оператор сложения  
 friend istream& operator>>(istream& is, Matrix& matrix); // Оператор ввода  
 friend ostream& operator<<(ostream& os, const Matrix& matrix); // Оператор вывода  
 friend ofstream& operator<<(ofstream& ofs, const Matrix& matrix); // Оператор вывода в файл  
 friend ifstream& operator>>(ifstream& ifs, Matrix& matrix); // Оператор ввода из файла  
 int\* operator[](int i) const; // Оператор доступа по индексу  
 void saveToBinary(const string& filename) const; // Сохранение в бинарный файл  
 void loadFromBinary(const string& filename); // Загрузка из бинарного файла  
 void print() const; // Печать матрицы  
 void resize(int newRows, int newCols); // Изменение размеров матрицы  
protected:  
 int rows; // Количество строк  
 int cols; // Количество столбцов  
 int\*\* data; // Данные матрицы  
private:  
 void freeMemory(); // Освобождение памяти  
};  
  
// Класс для однострочной матрицы  
class SingleMatrix : public Matrix {  
public:  
 SingleMatrix() : Matrix() {} // Конструктор по умолчанию  
 SingleMatrix(int rows, int cols) : Matrix(rows, cols) {} // Конструктор с заданием размеров  
 SingleMatrix(const SingleMatrix& other) : Matrix(other) {} // Конструктор копирования  
 ~SingleMatrix() {} // Деструктор  
};  
  
// Класс для прямоугольной матрицы  
class RectangularMatrix : public Matrix {  
public:  
 RectangularMatrix() : Matrix() {} // Конструктор по умолчанию  
 RectangularMatrix(int rows, int cols) : Matrix(rows, cols) {} // Конструктор с заданием размеров  
 RectangularMatrix(const SingleMatrix& other) : Matrix(other) {} // Конструктор из однострочной матрицы  
 ~RectangularMatrix() {} // Деструктор  
};  
  
#endif // MATRIX\_H

**Matrix.cpp**

#include "matrix.h"  
#include <iostream>  
#include <limits>  
#include <stdexcept>  
using namespace std;  
  
// Конструктор по умолчанию  
Matrix::Matrix() : rows(0), cols(0), data(nullptr) {}  
  
// Конструктор с параметрами для создания матрицы заданного размера  
Matrix::Matrix(int rows, int cols) : rows(rows), cols(cols) {  
 data = new int\* [rows]; // Выделение памяти для строк  
 for (int i = 0; i < rows; i++) {  
 data[i] = new int[cols]; // Выделение памяти для столбцов  
 }  
}  
  
// Конструктор копирования для копирования существующей матрицы  
Matrix::Matrix(const Matrix& other) : rows(other.rows), cols(other.cols) {  
 data = new int\* [rows];  
 for (int i = 0; i < rows; i++) {  
 data[i] = new int[cols];  
 for (int j = 0; j < cols; j++) {  
 data[i][j] = other.data[i][j]; // Копирование данных  
 }  
 }  
}  
  
// Деструктор для освобождения памяти  
Matrix::~Matrix() {  
 for (int i = 0; i < rows; ++i) {  
 delete[] data[i]; // Освобождение памяти для каждой строки  
 }  
 delete[] data; // Освобождение памяти для массива строк  
}  
  
// Метод для освобождения памяти  
void Matrix::freeMemory() {  
 if (data) {  
 for (int i = 0; i < rows; i++) {  
 delete[] data[i]; // Освобождение каждой строки  
 }  
 delete[] data; // Освобождение массива строк  
 data = nullptr; // Установка указателя в nullptr  
 }  
}  
  
// Метод для умножения матриц  
Matrix Matrix::multiply(const Matrix& other) const {  
 Matrix result(rows, other.cols); // Создание матрицы для результата  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < other.cols; j++) {  
 result.data[i][j] = 0; // Инициализация элемента результата  
 for (int k = 0; k < cols; k++) {  
 result.data[i][j] += data[i][k] \* other.data[k][j]; // Умножение  
 }  
 }  
 }  
 return result; // Возврат результата  
}  
  
// Метод для транспонирования матрицы  
Matrix Matrix::transpose() const {  
 Matrix result(cols, rows); // Создание матрицы для транспонированного результата  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 result.data[j][i] = data[i][j]; // Транспонирование  
 }  
 }  
 return result; // Возврат транспонированной матрицы  
}  
  
// Перегруженная операция присваивания  
Matrix Matrix::operator=(const Matrix& other) {  
 if (this != &other) {  
 freeMemory(); // Освобождение текущей памяти  
 rows = other.rows;  
 cols = other.cols;  
 data = new int\* [rows];  
 for (int i = 0; i < rows; i++) {  
 data[i] = new int[cols];  
 for (int j = 0; j < cols; j++) {  
 data[i][j] = other.data[i][j]; // Копирование данных  
 }  
 }  
 }  
 return \*this; // Возврат текущего объекта  
}  
  
// Перегруженная операция вычитания  
Matrix Matrix::operator-(const Matrix& other) const {  
 Matrix result(rows, cols); // Создание матрицы для результата  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 result.data[i][j] = data[i][j] - other.data[i][j]; // Вычитание  
 }  
 }  
 return result; // Возврат результата  
}  
  
// Перегруженная операция сложения  
Matrix operator+(const Matrix& m1, const Matrix& m2) {  
 Matrix result(m1.rows, m1.cols); // Создание матрицы для результата  
 for (int i = 0; i < m1.rows; i++) {  
 for (int j = 0; j < m1.cols; j++) {  
 result.data[i][j] = m1.data[i][j] + m2.data[i][j]; // Сложение  
 }  
 }  
 return result; // Возврат результата  
}  
  
// Перегрузка оператора ввода для матриц  
istream& operator>>(istream& is, Matrix& matrix) {  
 cout << "Add matrix numbers " << matrix.rows << "x" << matrix.cols << ":" << endl;  
 for (int i = 0; i < matrix.rows; i++) {  
 for (int j = 0; j < matrix.cols; j++) {  
 while (true) {  
 int value;  
 is >> value; // Ввод значения  
 if (is.fail()) {  
 is.clear(); // Сброс состояния  
 is.ignore(numeric\_limits<streamsize>::max(), '\n'); // Игнорируем некорректный ввод  
 cout << "Invalid input, please enter a number: "; // Сообщение об ошибке  
 } else {  
 matrix.data[i][j] = value; // Запись значения в матрицу  
 break;  
 }  
 }  
 }  
 }  
 return is; // Возврат потока  
}  
  
// Перегрузка оператора вывода для матриц  
ostream& operator<<(ostream& os, const Matrix& matrix) {  
 for (int i = 0; i < matrix.rows; i++) {  
 for (int j = 0; j < matrix.cols; j++) {  
 os << matrix.data[i][j] << " "; // Вывод элементов матрицы  
 }  
 os << endl; // Переход на новую строку  
 }  
 return os; // Возврат потока  
}  
  
// Сохранение матрицы в текстовый файл  
ofstream& operator<<(ofstream& ofs, const Matrix& matrix) {  
 ofs << matrix.rows << " " << matrix.cols << endl; // Запись размеров матрицы  
 for (int i = 0; i < matrix.rows; i++) {  
 for (int j = 0; j < matrix.cols; j++) {  
 ofs << matrix.data[i][j] << " "; // Запись элементов матрицы  
 }  
 ofs << endl; // Переход на новую строку  
 }  
 return ofs; // Возврат потока  
}  
  
// Загрузка матрицы из текстового файла  
ifstream& operator>>(ifstream& ifs, Matrix& matrix) {  
 ifs >> matrix.rows >> matrix.cols; // Чтение размеров матрицы  
 matrix.resize(matrix.rows, matrix.cols); // Изменение размера матрицы  
 for (int i = 0; i < matrix.rows; ++i) {  
 for (int j = 0; j < matrix.cols; ++j) {  
 ifs >> matrix.data[i][j]; // Чтение данных в матрицу  
 }  
 }  
 return ifs; // Возврат потока  
}  
  
// Метод для сохранения матрицы в бинарный файл  
void Matrix::saveToBinary(const string& filename) const {  
 ofstream ofs(filename, ios::binary); // Открытие файла для записи в бинарном режиме  
 if (ofs.is\_open()) {  
 ofs << \*this; // Используем перегруженный оператор для записи  
 ofs.close(); // Закрытие файла  
 }  
}  
  
// Метод для загрузки матрицы из бинарного файла  
void Matrix::loadFromBinary(const string& filename) {  
 ifstream ifs(filename, ios::binary); // Открытие файла для чтения в бинарном режиме  
 if (ifs.is\_open()) {  
 ifs >> \*this; // Используем перегруженный оператор для чтения  
 ifs.close(); // Закрытие файла  
 }  
}  
  
// Печать матрицы  
void Matrix::print() const {  
 for (int i = 0; i < rows; i++) {  
 for (int j = 0; j < cols; j++) {  
 cout << data[i][j] << " "; // Вывод каждого элемента  
 }  
 cout << endl; // Переход на новую строку  
 }  
}  
  
// Метод для изменения размера матрицы  
void Matrix::resize(int newRows, int newCols) {  
 freeMemory(); // Освобождение текущей памяти  
 rows = newRows; // Установка нового количества строк  
 cols = newCols; // Установка нового количества столбцов  
 data = new int\*[rows]; // Выделение памяти для строк  
 for (int i = 0; i < rows; i++) {  
 data[i] = new int[cols](); // Выделение памяти для столбцов с инициализацией  
 }  
}