**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСТИЕТ**

Лабораторная работа №3 по дисциплине «Программирование»

**Организация ввода-вывода в С++**

Группа: **АВТ-342**

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

НОВОСИБИРСК 2024

**1.Постановка задачи**

**Вариант 4.**

Для класса из лаб. работы №2 перегрузить операции ввода/вывода, позволяющие осуществлять ввод и вывод в удобной фоpме объектов классов:

* ввод с клавиатуры объекта и вывод на экран;
* запись объектов в текстовый файл и загрузка из текстового файла;
* запись объектов в двоичный файл и загрузка из двоичного файла.

Изменить демонстpационную пpогpамму. В файл иметь возможность сохранять любое количество созданных объектов. При загрузке в программу создавать объекты с параметрами, записанными в файл.

**2. Определение пользовательского класса**

Класс Matrix предназначен для работы с квадратными матрицами и включает в себя методы для выполнения базовых арифметических операций и работу с файлами.

**Основные элементы класса**:

* Поле int size для хранения размера матрицы.
* Динамический двумерный массив int\*\* data для хранения элементов матрицы.
* Методы для арифметических операций, транспонирования, сохранения и загрузки матриц.

**Методы класса**:

* Конструкторы (по умолчанию, с параметром и копирования).
* Деструктор для освобождения памяти.
* Методы для выполнения операций: сложения, вычитания, умножения и транспонирования.
* Методы для работы с файлами (сохранение и загрузка в текстовый и бинарный формат).

**3. Реализация перегруженных операций**

**Операция сложения (operator+)**:

* Реализована как дружественная функция. Это позволяет реализовать её вне класса, но с доступом к приватным данным объектов.
* Данная операция принимает два объекта класса Matrix и возвращает новую матрицу, являющуюся результатом сложения двух исходных матриц.

**Операция вычитания (operator-)**:

* Реализована как метод-член класса. Это делает её более естественной при обращении к методам объекта (например, matrix1 - matrix2).

**Операция умножения (multiply)**:

* Реализована как метод-член класса, так как для выполнения операции нужны данные обоих объектов, и это позволяет напрямую обращаться к их элементам.

**Операторы ввода/вывода** (operator>> и operator<<)\*\*:

* Реализованы как дружественные функции, что позволяет перегрузить их для стандартных потоков ввода/вывода и файловых потоков. Это даёт возможность корректно работать с объектами класса при вводе и выводе данных в удобном для пользователя виде.

**Запись и чтение матрицы в/из текстового и бинарного файла**В классе Matrix реализованы функции для сохранения матрицы в текстовом и бинарном формате, а также для загрузки матрицы из этих файлов. Эти функции позволяют сохранять текущие данные матрицы на диск и затем загружать их обратно, обеспечивая удобство работы с матрицами и возможность их использования в последующих сессиях программы.  
**Запись матрицы в текстовый файл**Запись матрицы в текстовый файл позволяет сохранить её в формате, который можно открыть и прочитать как обычный текстовый файл.

Код в matrix.h:  
friend std::ofstream& operator<<(std::ofstream& ofs, const Matrix& matrix);  
Реализация в matrix.cpp:  
ofstream& operator<<(ofstream& ofs, const Matrix& matrix) {  
ofs << matrix.size << endl; // Сначала записываем размер матрицы.  
for (int i = 0; i < matrix.size; i++) {  
for (int j = 0; j < matrix.size; j++) {  
ofs << [matrix.data](https://web.telegram.org/a/matrix.data)[i][j] << " "; // Записываем элементы матрицы.  
}  
ofs << endl;  
}  
return ofs;  
}:  
В файл сначала записывается размер матрицы size.  
Затем данные матрицы записываются в виде строк, где каждая строка соответствует одной строке матрицы.  
Эти данные можно прочитать человеком или программой, поддерживающей текстовый формат, и использовать для восстановления матрицы.  
**Загрузка матрицы из текстового файла**Для загрузки матрицы из текстового файла реализована перегруженная функция operator>>, которая позволяет считывать матрицу, записанную в текстовом виде.

matrix.h:  
friend std::ifstream& operator>>(std::ifstream& ifs, Matrix& matrix);

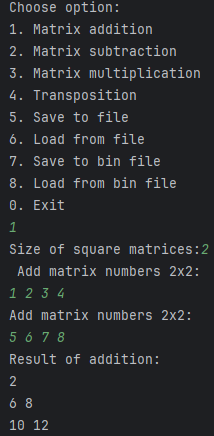
matrix.cpp:  
ifstream& operator>>(ifstream& ifs, Matrix& matrix) {  
ifs >> matrix.size; // Считываем размер матрицы.  
matrix.resize(matrix.size); // Изменяем размер матрицы.  
for (int i = 0; i < matrix.size; i++) {  
for (int j = 0; j < matrix.size; j++) {  
ifs >> [matrix.data](https://web.telegram.org/a/matrix.data)[i][j]; // Считываем элементы матрицы.  
}  
}  
return ifs;  
}  
Сначала из файла считывается размер матрицы и вызывается метод resize(), чтобы изменить размер матрицы.  
После этого последовательно считываются элементы матрицы и размещаются в её массиве data.  
Таким образом, матрица восстанавливается в том виде, в каком была сохранена.  
**Запись матрицы в бинарный файл**Запись в бинарный файл позволяет сохранить данные в двоичном формате, который более компактен и быстрее читается/записывается, хотя его нельзя просмотреть в текстовом виде.

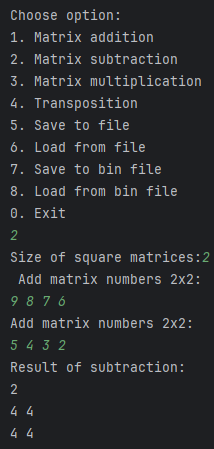
Код в matrix.h:  
void saveToBinary(const std::string& filename) const;

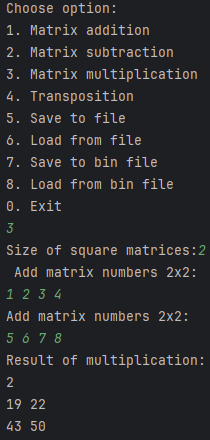
Реализация в matrix.cpp:  
void Matrix::saveToBinary(const std::string& filename) const {  
ofstream ofs(filename, ios::binary); // Открываем файл в двоичном режиме.  
ofs.write((char\*)&size, sizeof(size)); // Записываем размер матрицы.  
for (int i = 0; i < size; i++) {  
ofs.write((char\*)data[i], sizeof(int) \* size); // Записываем каждую строку матрицы.  
}  
ofs.close();  
}  
Сначала в файл записывается размер матрицы size как целое число.  
Затем каждая строка матрицы записывается как блок данных типа int, что позволяет сохранить всю матрицу быстро и компактно.  
В результате создается файл с двоичным представлением данных, который нельзя просмотреть как текст.  
**Загрузка матрицы из бинарного файла**Загрузка матрицы из бинарного файла восстанавливает её из двоичного представления, записанного функцией saveToBinary.  
Код в matrix.h:  
void loadFromBinary(const std::string& filename);

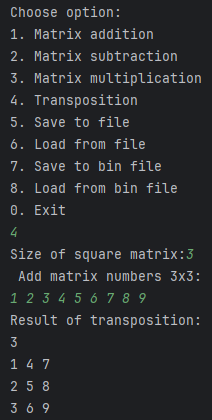
Реализация в matrix.cpp:  
void Matrix::loadFromBinary(const std::string& filename) {  
ifstream ifs(filename, ios::binary); // Открываем файл в двоичном режиме.  
[ifs.read](https://web.telegram.org/a/ifs.read)((char\*)&size, sizeof(size)); // Считываем размер матрицы.  
resize(size); // Изменяем размер матрицы.  
for (int i = 0; i < size; i++) {  
[ifs.read](https://web.telegram.org/a/ifs.read)((char\*)data[i], sizeof(int) \* size); // Считываем каждую строку матрицы.  
}  
ifs.close();  
}  
Из файла считывается размер матрицы size.  
Вызывается метод resize(), чтобы выделить память для матрицы заданного размера.  
Каждая строка матрицы считывается как блок данных и восстанавливается в массиве data.  
В результате матрица полностью восстанавливается в исходном виде.

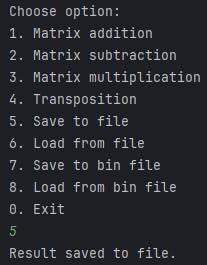
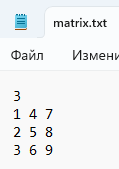
**4. Тестовые данные и результаты тестирования**

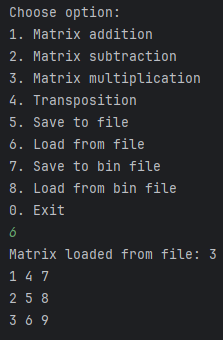
Тест сложения матриц:

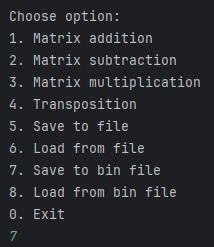
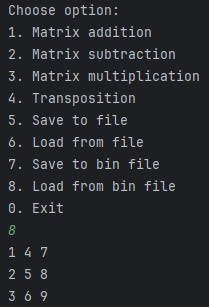
Тест вычитания матриц:

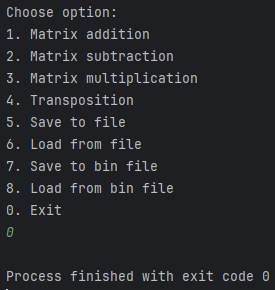
Тест умножения матриц:

Тест транспонирования матрицы:

Тест сохранения в текстовый файл:

Тест загрузки матрицы из текстового файла:

Тест сохранения матрицы в бинарный файл:  
Тест загрузки матрицы из бинарного файла:  
****

Тест выхода из консоли:

Тесты проведены, программа работает правильно.  
**Вывод**В ходе выполнения лабораторной работы была разработана программа для выполнения различных операций над квадратными матрицами, включая сложение, вычитание, умножение, транспонирование, а также сохранение и загрузку данных матрицы в текстовый и бинарный файлы.  
Была реализована структура класса Matrix, которая предоставляет методы для выполнения математических операций и работы с файлами. Также были перегружены операторы, такие как сложение, вычитание, ввод и вывод, что делает работу с матрицами более удобной и интуитивной.  
Для хранения данных использовались как текстовые, так и бинарные файлы. Бинарный формат позволил сократить объем сохраняемой информации и ускорить процессы записи и чтения, тогда как текстовый формат предоставил возможность визуального контроля данных в файле.  
Программа успешно прошла тестирование на примерах различных операций и корректно выполняет задачи, поставленные в рамках задания.  
**5. 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 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 size;

switch (choice) {

case 1: { // Сложение матриц

cout << "Size of square matrices: ";

cin >> size;

Matrix matrix1(size), matrix2(size);

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: ";

cin >> size;

Matrix matrix1(size), matrix2(size);

cin >> matrix1 >> matrix2;

result = matrix1 - matrix2;

cout << "Result of subtraction:" << endl << result;

is\_result\_stored = true;

break;

}

case 3: { // Умножение матриц

cout << "Size of square matrices: ";

cin >> size;

Matrix matrix1(size), matrix2(size);

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 square matrix: ";

cin >> size;

Matrix matrix1(size);

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: " << 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;

}

}

return 0;

}

matrix.h:  
#ifndef MATRIX\_H

#define MATRIX\_H

#include <iostream>

#include <fstream>

using namespace std;

class Matrix {

public:

Matrix();

Matrix(int size);

Matrix(const Matrix&);

~Matrix();

void input();

void output() const;

Matrix multiply(const Matrix&) const;

Matrix transpose() const;

int determinant() const;

Matrix operator=(const Matrix&);

Matrix operator-(const Matrix&) const;

int operator()() const;

int& operator()(int row, int column);

// Объявления перегруженных операторов

friend Matrix operator+(const Matrix&, const Matrix&);

friend std::istream& operator>>(std::istream& is, Matrix& matrix);

friend std::ostream& operator<<(std::ostream& os, const Matrix& matrix);

// Перегруженные операторы для бинарных файлов

friend std::ofstream& operator<<(std::ofstream& ofs, const Matrix& matrix);

friend std::ifstream& operator>>(std::ifstream& ifs, Matrix& matrix);

int getSize() const;

void resize(int newSize);

int\* operator[](int i) const;

void saveToBinary(const std::string& filename) const;

void loadFromBinary(const std::string& filename);

void print() const;

private:

int size;

int\*\* data;

void freeMemory();

};

#endif

matrix.cpp:  
#include "matrix.h"

#include <cmath>

#include <iostream>

#include <limits>

#include <stdexcept>

using namespace std;

// Конструктор по умолчанию

Matrix::Matrix() : size(0), data(nullptr) {}

// Конструктор с параметром

Matrix::Matrix(int size) : size(size) {

data = new int\* [size];

for (int i = 0; i < size; i++) {

data[i] = new int[size];

}

}

// Конструктор копирования

Matrix::Matrix(const Matrix& other) : size(other.size) {

data = new int\* [size];

for (int i = 0; i < size; i++) {

data[i] = new int[size];

for (int j = 0; j < size; j++) {

data[i][j] = other.data[i][j];

}

}

}

// Деструктор

Matrix::~Matrix() {

freeMemory();

}

// Освобождение памяти

void Matrix::freeMemory() {

if (data) {

for (int i = 0; i < size; i++) {

delete[] data[i];

}

delete[] data;

data = nullptr;

}

}

// Операция умножения

Matrix Matrix::multiply(const Matrix& other) const {

Matrix result(size);

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

result.data[i][j] = 0;

for (int k = 0; k < size; k++) {

result.data[i][j] += data[i][k] \* other.data[k][j];

}

}

}

return result;

}

// Транспонирование матрицы

Matrix Matrix::transpose() const {

Matrix result(size);

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

result.data[i][j] = data[j][i];

}

}

return result;

}

// Перегруженная операция присваивания

Matrix Matrix::operator=(const Matrix& other) {

if (this != &other) {

freeMemory();

size = other.size;

data = new int\* [size];

for (int i = 0; i < size; i++) {

data[i] = new int[size];

for (int j = 0; j < size; j++) {

data[i][j] = other.data[i][j];

}

}

}

return \*this;

}

// Перегруженная операция вычитания

Matrix Matrix::operator-(const Matrix& other) const {

Matrix result(size);

for (int i = 0; i < size; i++) {

for (int j = 0; j < size; j++) {

result.data[i][j] = data[i][j] - other.data[i][j];

}

}

return result;

}

// Перегруженная операция сложения

Matrix operator+(const Matrix& m1, const Matrix& m2) {

Matrix result(m1.size);

for (int i = 0; i < m1.size; i++) {

for (int j = 0; j < m1.size; j++) {

result.data[i][j] = m1.data[i][j] + m2.data[i][j];

}

}

return result;

}

// Перегрузка оператора ввода для матриц

std::istream& operator>>(std::istream& is, Matrix& matrix) {

std::cout << "Add matrix numbers " << matrix.size << "x" << matrix.size << ":" << std::endl;

for (int i = 0; i < matrix.size; i++) {

for (int j = 0; j < matrix.size; j++) {

while (true) {

int value;

is >> value;

if (is.fail()) {

is.clear(); // Сброс состояния

is.ignore(numeric\_limits<streamsize>::max(), '\n'); // Игнорируем некорректный ввод

std::cout << "Invalid input, please enter a number: ";

} else {

matrix.data[i][j] = value;

break;

}

}

}

}

return is;

}

// Перегрузка оператора вывода для матриц

std::ostream& operator<<(std::ostream& os, const Matrix& matrix) {

os << matrix.size << std::endl; // Вывод размера матрицы

for (int i = 0; i < matrix.size; i++) {

for (int j = 0; j < matrix.size; j++) {

os << matrix.data[i][j] << " ";

}

os << std::endl;

}

return os;

}

// Сохранение матрицы в бинарный файл

std::ofstream& operator<<(std::ofstream& ofs, const Matrix& matrix) {

ofs.write(reinterpret\_cast<const char\*>(&matrix.size), sizeof(matrix.size)); // Записываем размер матрицы

for (int i = 0; i < matrix.size; ++i) {

ofs.write(reinterpret\_cast<const char\*>(matrix.data[i]), matrix.size \* sizeof(int)); // Записываем данные

}

return ofs;

}

// Загрузка матрицы из бинарного файла

std::ifstream& operator>>(std::ifstream& ifs, Matrix& matrix) {

ifs.read(reinterpret\_cast<char\*>(&matrix.size), sizeof(matrix.size)); // Считываем размер матрицы

matrix.resize(matrix.size); // Изменяем размер матрицы

for (int i = 0; i < matrix.size; ++i) {

ifs.read(reinterpret\_cast<char\*>(matrix.data[i]), matrix.size \* sizeof(int)); // Считываем данные

}

return ifs;

}

// Метод для сохранения матрицы в бинарный файл

void Matrix::saveToBinary(const std::string& filename) const {

std::ofstream ofs(filename, std::ios::binary);

if (ofs.is\_open()) {

ofs << \*this; // Используем перегруженный оператор для записи

ofs.close();

}

}

// Метод для загрузки матрицы из бинарного файла

void Matrix::loadFromBinary(const std::string& filename) {

std::ifstream ifs(filename, std::ios::binary);

if (ifs.is\_open()) {

ifs >> \*this; // Используем перегруженный оператор для чтения

ifs.close();

}

}

// Печать матрицы

void Matrix::print() const {

for (int i = 0; i < size; ++i) {

for (int j = 0; j < size; ++j) {

cout << data[i][j] << " ";

}

cout << endl;

}

}

// Функция изменения размера матрицы

void Matrix::resize(int newSize) {

freeMemory();

size = newSize;

data = new int\* [size];

for (int i = 0; i < size; i++) {

data[i] = new int[size]();

}

}