**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСТИЕТ**

Лабораторная работа №2 по дисциплине «Программирование»

**Переопределение операций**

Группа: **АВТ-342**

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

НОВОСИБИРСК 2024

**1.** **Постановка задачи**

**Вариант 4**

Для класса из лаб. работы №1 реализовать набор операций для работы с объектами класса: по заданию. Изменить демонстpационную пpогpамму, продемострировав все перегруженные операции. Операции перегрузить методами класса и дружественными функциями.

Для класса pаботы с квадратными матрицами перегрузить следующие операции:

* сложение/ вычитание матриц;
* операцию индексирования;
* операцию вызова функции – как получение определителя;
* операцию присваивания.

**2. Определение класса**

Имея код из лабораторной работы №1, класс Matrix в matrix.h остается неизменным, за исключением новых функций для перегрузки:

* friend Matrix operator+ (const Matrix&, const Matrix&) - дружественная функция для перегруза операции сложения;
* Matrix operator- (const Matrix&) - функция-член для перегруза операции вычитания;
* Matrix operator= (const Matrix&) - функция-член для перегруза операции присваивания;
* int operator()() const - функция-член для перегруза операции вызова функции – как получения определителя;
* int& operator() (int row, int column) - функция-член для перегруза операции индексации, где row – строка, column – столбец матрицы.

**3. Обоснование включения в класс конструкторов, деструктора и операции присваивания**

* **Конструкторы** были включены в класс Matrix еще на этапе проектирования программы при выполнении лабораторной работы №1, где требовлось наличие конструктора по умолчанию, параметрами и копирования. Так как в задании требуется дополнить код с предыдущей работы, конструкторы остаются неизменными, однако нам понадобится только конструктор с параметрами.
* **Деструктор** отвечает за освобождение динамически выделенной памяти для хранения элементов матрицы.
* **Операция присваивания** (Matrix::operator=(const Matrix& other)) позволяет корректно копировать значения одной матрицы в другую, включая выделение новой памяти, если это необходимо. Объект проверяет, не является ли он самоприсваиваемым, и освобождает старую память перед присваиванием новых данных.

**4. Реализация перегруженных операций с обоснованием выбранного способа**

* **Операция сложения** реализована как **дружественная функция**, однако в коде есть возможность реализовать ее с помощью функции-члена, так как функция расположена в классе Matrix. Преимущество дружественной функции заключается в том, что она позволяет получить доступ к приватным полям обоих объектов матриц (размер и данные) и обеспечивать возможность использовать оператор сложения с двумя объектами класса Matrix, находясь вне класса.

Перегруз операции сложения проходит по аналогичной формуле сложения в matrix.cpp, однако мы передаем не ссылку на экземпляр класса, а сам экземпляр. Соответственно, сложение выполняется, взаимодействуя с индексами матрицы result.data[i][j] = m1.data[i][j] + m2.data[i][j].

* **Операция вычитания** реализована как **функция-член**. Данный способ реализации выбран в целях удобства и экономии места. Операция находится внутри класса, соответственно не требует внешнего доступа к двум объектам, что удобно для вызова через объект.

Перегруз операции вычитания проходит аналогично сложению: происходит взаимодействие с объектами. Значит, в main.cpp результат выглядет следующим образом: Matrix result = matrix1 - matrix2.

* **Операция вызова функции – как получение определителя** реализована как функция член, по аналогичным причинам с операцией вычитания.

Так как требуется вызов функции для получения определителя, перегрузка будет возращать метод determinant.

* **Операция индексирования** реализована как функция член и перегружена для доступа к элементам матрицы через индексы, что упрощает доступ к элементам матрицы с использованием удобного синтаксиса.

Нам необходимо индексирование, то есть поиск числа по его индексу и редактирование. Перегруз осуществляется через проход по всей матрицы с индексами строки и столба. Если был произведен выход за границы в матрице, сообщение об ошибке выведется в блоке try, а именно с помощью catch.

* **Операция присваивания** реализована как функция-член. Кроме того, данная операция не может быть реализована как дружественная функция.

Аналогично с остальными перегрузами функций операций, операция присваивания проходит с обращением к объектам напрямую. В самом алгоритме элементы первой матрицы присваиваются второй поочередно.

**5. Текст программы с комментариями**

matrix.h

#ifndef MATRIX\_H

#define MATRIX\_H

#include <iostream>

class Matrix {

public:

static int method\_cnt;

Matrix(); // Конструктор по умолчанию

Matrix(int size); // Конструктор с параметром

Matrix(const Matrix&); // Конструктор копирования

~Matrix(); // Деструктор

void input();

void output() const;

Matrix multiply(const Matrix&) const;

Matrix transpose() const;

int determinant() const;

static int getcount();

Matrix operator=(const Matrix&);

Matrix operator-(const Matrix&) const;

int operator()() const;

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

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

private:

int size;

int\*\* data;

void freeMemory();

};

#endif

matrix.cpp

#include "matrix.h"

#include <cmath>

using namespace std;

int Matrix::method\_cnt = 0;

// 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() {

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

delete[] data[i];

}

delete[] data;

}

// Ввод матрицы

void Matrix::input() {

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

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

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

while (true) {

int value;

cin >> value;

if (cin.fail()) {

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "n/a" << endl;

exit(1);

}

else {

data[i][j] = value;

break;

}

}

}

}

}

// Вывод матрицы

void Matrix::output() const {

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

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

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

}

cout << endl;

}

}

// Операция умножения

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

method\_cnt++;

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 {

method\_cnt++;

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;

}

// Вычисление детерминанта

int Matrix::determinant() const {

method\_cnt++;

if (size == 1) {

return data[0][0];

}

if (size == 2) {

return (data[0][0] \* data[1][1] - data[1][0] \* data[0][1]);

}

if (size == 3) {

return ((data[0][0] \* data[1][1] \* data[2][2]) + (data[0][1] \* data[1][2] \* data[2][0])

+ (data[0][2] \* data[1][0] \* data[2][1]) - (data[0][2] \* data[1][1] \* data[2][0]) -

(data[0][1] \* data[1][0] \* data[2][2]) - (data[0][0] \* data[1][2] \* data[2][1]));

}

else {

int det = 0;

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

Matrix submatrix(size - 1);

int subi = 0;

for (int i = 1; i < size; i++) {

int subj = 0;

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

if (j == x) continue;

submatrix.data[subi][subj] = data[i][j];

subj++;

}

subi++;

}

det += (pow(-1, x) \* data[0][x] \* submatrix.determinant());

}

return det;

}

}

//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 {

method\_cnt++;

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;

}

//Перегруженный вызов определителя

int Matrix::operator()() const {

return determinant();

}

//Перегруженная операция индексирования

int& Matrix::operator()(int row, int column) {

if (row >= 0 && row < size && column >= 0 && column < size) {

return data[row][column];

}

else {

throw std::out\_of\_range("Out of range");

}

}

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;

}

main.cpp

#include "matrix.h"

#include <iostream>

using namespace std;

int main() {

int size, 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. Determinant" << endl;

cout << "6. Indexsation" << endl;

cout << "7. Assigment" << endl;

cin >> choice;

switch (choice) {

case 1: {

cout << "Size of square matrices: ";

cin >> size;

Matrix matrix1(size);

matrix1.input();

Matrix matrix2(size);

matrix2.input();

Matrix result = matrix1 + matrix2;

cout << "Result of addition:" << endl;

result.output();

break;

}

case 2: {

cout << "Size of square matrices: ";

cin >> size;

Matrix matrix1(size);

matrix1.input();

Matrix matrix2(size);

matrix2.input();

Matrix result = matrix1 - matrix2;

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

result.output();

break;

}

case 3: {

cout << "Size of square matrices: ";

cin >> size;

Matrix matrix1(size);

matrix1.input();

Matrix matrix2(size);

matrix2.input();

Matrix result = matrix1.multiply(matrix2);

cout << "Result of multiplication:" << endl;

result.output();

break;

}

case 4: {

cout << "Size of square matrix: ";

cin >> size;

Matrix matrix1(size);

matrix1.input();

Matrix result = matrix1.transpose();

cout << "Result of transposition:" << endl;

result.output();

break;

}

case 5: {

cout << "Size of square matrix: ";

cin >> size;

Matrix matrix1(size);

matrix1.input();

int det = matrix1();

cout << "Determinant of the matrix: " << det << endl;

break;

}

case 6: {

cout << "Size of square matrix:";

cin >> size;

Matrix matrix1(size);

matrix1.input();

int row, column;

cout << "Enter the row index: ";

cin >> row;

cout << "Enter the column index: ";

cin >> column;

try { //Если найдена ошибка, то catch “ловит и выводит ее”

int value = matrix1(row, column);

cout << "Value at (" << row << ", " << column << "): " << value << endl;

cout << "Enter new value for this element: ";

cin >> value;

matrix1(row, column) = value;

cout << "Updated matrix: " << endl;

matrix1.output();

}

catch (const std::out\_of\_range& e) {

cout << e.what() << endl;

}

break;

}

case 7: {

cout << "Size of square matrices: ";

cin >> size;

Matrix matrix1(size);

matrix1.input();

Matrix matrix2(size);

matrix2 = matrix1;

cout << "Matrix 2:" << endl;

matrix2.output();

break;

}

default:

cout << "Invalid option" << endl;

break;

}

// Вывод количества вызванных методов

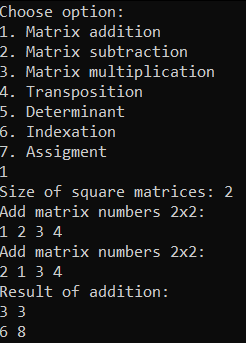
cout << "Number of matrix methods used: " << Matrix::method\_cnt << endl;

return 0;

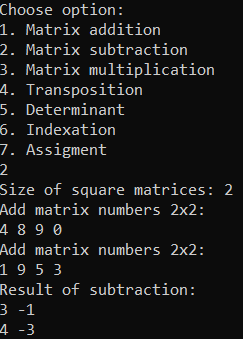
}

**6. Тестовые данные и результаты тестирования**

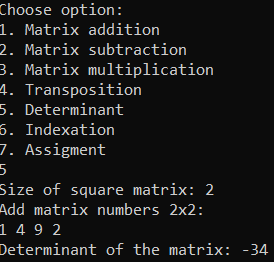
**1) Сложение матриц**



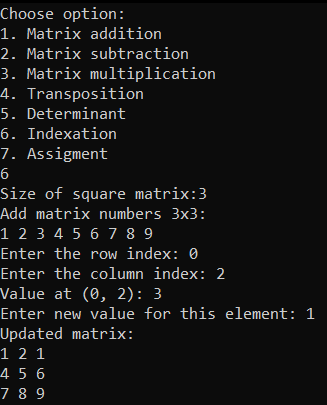
**2) Вычитание матриц**



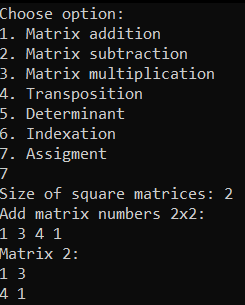
**3) Нахождение определителя**



**4) Индексация**

****

**5) Присваивание**

****

**7. Вывод**

В ходе выполнения лабораторной работы, мы ознакомились с особенностями использования дружественных классов и функций, а также возможностью получения законченного нового типа данных, определив для него допустимые операции с помощью перегрузки операторов. Кроме того, в программе была реализована перегрузка оператций сложения, вычитания, обращения к функции – нахождения определителя, индексирования и присваивания матриц с использованием функций-членов и дружественных функций.