МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра автоматизированных систем управления

Отчет по лабораторной работе №2.

По ОБЪЕКТО-ОРИЕНТИРОВАННОМУ ПРОГРАММИРОВАНИЮ

«Конструкторы и деструктор. Перегрузка операторов»

Выполнил:

студент гр. ПМ-20-2 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Бутусова В.М.

Проверил:

доц., к.п.н. кафедры АСУ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кургасов В. В.

Липецк 2022

**Цель работы:**

Изучить виды конструкторов, принципы использования конструкторов, назначение деструктора, способы перегрузки операторов.

**Задание кафедры:**

Реализовать на языке С++ класс, который обладает возможностью выполнения соответствующих действий. Реализованный класс должен содержать необходимые конструкторы, деструктор и перегруженные операторы.

**Вариант 3:**

Умножение двух матриц и присвоение результата третьей переменной-матрице осуществляется одной строкой C=A\*B.

**Исходный код:**

#include <iostream>

using namespace std;

class Matrica

{

private:

int c1, c2;

float\*\* mas; //объявление массива и размерности

public:

Matrica()

{

c1 = 0;

c2 = 0;

mas = 0;

}

Matrica(float\*\* massiv, int n1, int n2)//конструктор с параметрами

{ //создаёт обрабатываемую матрицу

{

c1 = n1;

c2 = n2;

mas = new float\* [n1];

for (int i = 0; i < n1; i++)

{

mas[i] = new float[n2];

}

for (int i = 0; i < n1; i++)

{

for (int j = 0; j < n2; j++)

{

mas[i][j] = massiv[i][j];

}

}

}

}

Matrica(const Matrica& matricaA) // Конструктор копирования

{

c1 = matricaA.c1;

c2 = matricaA.c2;

mas = new float\* [c1];

for (int i = 0; i < c1; i++)

{

for (int j = 0; j < c2; j++)

{

mas[i][j] = matricaA.mas[i][j];

}

}

}

friend Matrica operator \* (Matrica& B, Matrica& C);

Matrica& operator=(const Matrica& ravno) //оператор присваивания

{ // присваевает матрице C результат перемножения матриц A и B

this->c1 = ravno.c1;

this->c2 = ravno.c2;

this->mas = new float\* [c1];

for (int i = 0; i < c1; i++)

{

this->mas[i] = new float[c2];

}

for (int i = 0; i < c1; i++)

{

for (int j = 0; j < c2; j++)

{

this->mas[i][j] = ravno.mas[i][j];

}

}

return \*this;

}

void vivod(string stroka)//функция выводит матрицу

{

cout << endl << stroka << endl;

for (int i = 0; i < c1; i++)

{

for (int j = 0; j < c2; j++)

{

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

}

cout << endl;

}

}

~Matrica()//деструктор

{ // очищает массив

for (int i = 0; i < c2; i++)

{

delete[] mas[i];

}

delete[] mas;

}

};

Matrica operator\* (Matrica& A, Matrica& B) // Перегрузка оператора для умножения матрицы A на матрицу B

{

if (A.c2 == B.c1) //если число столбцов матрицы A == числу строк матрицы B

{ //производим умножение

float\*\* N = new float\* [A.c1];

for (int i = 0; i < A.c1; i++)

{

N[i] = new float[B.c2];

for (int j = 0; j < B.c2; j++)

{

N[i][j] = 0;

for (int k = 0; k < A.c2; k++)

{

N[i][j] += A.mas[i][k] \* B.mas[k][j];

}

}

}

Matrica new\_C(N, A.c1, B.c2); //для дальнейшего вывода матрицы на экран используем конструктор с параметрами

cout << endl << "Операция умножения";

new\_C.vivod("Матрица C:"); // вызываем функцию вывода матрицы

for (int i = 0; i < B.c2; i++)

{

delete[] N[i];

}

delete[] N;

return new\_C;

}

else //иначе сообщаем, что матрицы перемножить нельзя

{

int c1=0, c2=0;

float\*\* mas;

mas = 0;

Matrica new\_C(mas, c1, c2);

new\_C.vivod("Матрицы нельзя перемножить");

return new\_C;

}

}

int main()

{

setlocale(LC\_ALL, "rus");

int a1, a2, b1, b2;

float\*\* mas1;

float\*\* mas2;

cout << "Введите размерность матрицы A: ";

cin >> a1 >> a2;

mas1 = new float\* [a1];

for (int i = 0; i < a1; i++) //создание первого массива

{

mas1[i] = new float[a2];

}

cout << "Введите элементы матрицы A: " << endl;

for (int i = 0; i < a1; i++)

{

for (int j = 0; j < a2; j++)

{

float k;

cin >> k;

mas1[i][j] = k;

}

}

cout << "Введите размерность матрицы B: ";

cin >> b1 >> b2;

mas2 = new float\* [b1];

for (int i = 0; i < b1; i++) //создание второго массива

{

mas2[i] = new float[b2];

}

cout << "Введите элементы матрицы B: " << endl;

for (int i = 0; i < b1; i++)

{

for (int j = 0; j < b2; j++)

{

float k;

cin >> k;

mas2[i][j] = k;

}

}

Matrica C;

Matrica A(mas1, a1, a2);

Matrica B(mas2, b1, b2);

A.vivod("Матрица A");

B.vivod("Матрица B");

C = A \* B;

}

**Результаты работы программы:**

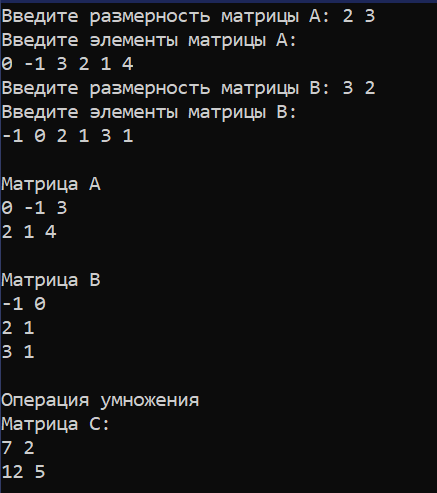
****

Рисунок 1 – перемножение матриц разных размерностей.

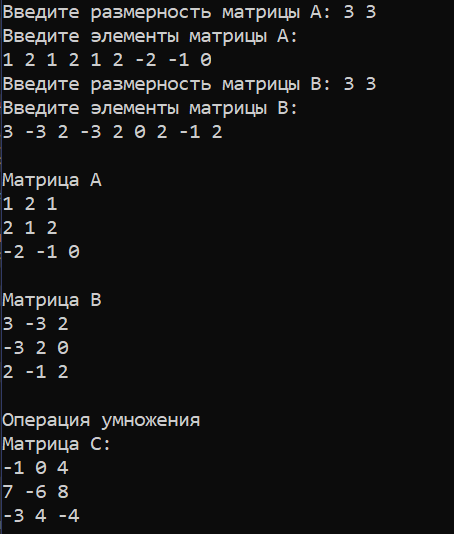


Рисунок 2 – перемножение квадратных матриц.

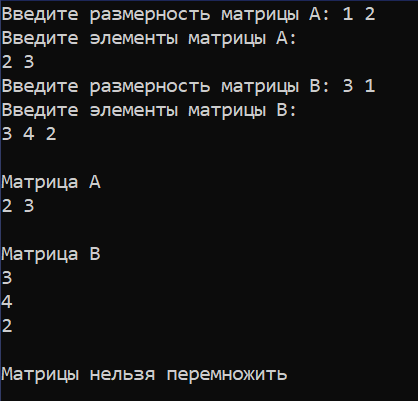


Рисунок 3 – исключение.

**Вывод:**

Я изучила виды конструкторов, принципы использования конструкторов, назначение деструктора, способы перегрузки операторов.

**Контрольные вопросы:**

**1. Что такое конструктор и деструктор класса?**

***Конструктор*** класса – это специальный метод (функция) класса. Конструктор вызывается при создании объекта класса. Как правило, конструктор используется для:

* выделения памяти под объект класса;
* начальной инициализации внутренних данных класса.

Конструктор предназначен для формирования экземпляра объекта класса. Имя конструктора класса совпадает с именем класса.

Деструктор — это функция-член, которая вызывается автоматически при выходе объекта из области действия или явно уничтожена вызовом **delete**.

**2. Какие бывают конструкторы?**

· Конструктор по умолчанию

· Конструктор с параметрами

· Конструктор копирования

· Конструктор преобразования

**3. Зачем нужен конструктор копирования?**

Конструктор копирования нужен нам для того, чтобы создавать «реальные» копии объектов класса, а не побитовую копию объекта.

* когда мы передаем объект в какую-либо функцию в виде параметра;
* когда какая-либо функция должна вернуть объект класса в результате своей работы;
* когда мы в главной функции один объект класса инициализируем другим объектом класса.

**4. Зачем нужен перегруженный оператор присваивания?**

Если в программе объявить класс, в котором оператор присваивания не перегружается, то для этого класса компилятором будет создан оператор присваивания по умолчанию. Итак, оператор присваивания по умолчанию – это такой оператор присваивания, который автоматически создается компилятором для обеспечения копирования одного экземпляра класса другому экземпляру класса. При вызове оператора присваивания по умолчанию происходит побайтовое копирование одного экземпляра класса другому. В таком случае экземпляры классов будут указывать на один и тот же участок памяти.

Если нужно присваивать экземпляры классов друг другу таким образом, чтобы они были не связаны между собой, то для этого в классе можно реализовать операторную функцию operator=(), которая будет выполнять соответствующие операции.

**5. Какие существуют способы перегрузки операторов для класса?**

Существует два основных способа перегрузки операторов: **глобальные функции, дружественные для класса, или подставляемые функции самого класса**.

**6. Когда срабатывает конструктор копирования (без явного вызова)?**

Неявный конструктор копирования срабатывает при передаче объекта в функцию.

**7. Когда вызывается деструктор класса?**

Деструкторы всегда вызываются при выходе объекта класса из области видимости али при их удалении оператором delete.