**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра Автоматизированных систем управления

Лабораторная работа № 2

по ОБЪЕКТНО-ОРИЕНТИРОВАННОМУ ПРОГРАММИРОВАНИЮ

«Конструкторы и деструктор. Перегрузка операторов»

Студент \_\_\_\_\_\_\_\_\_\_\_ Целищев А.Е.

Группа ПМ-21-2

Руководитель

Доцент \_\_\_\_\_\_\_\_\_\_\_ Кургасов В.В.

Липецк 2022г.

Оглавление

Цель работы 2

Задание кафедры 2

Код программы 3

Пример работы программы 6

Вывод 7

Контрольные вопросы...........................................................................................8

**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 перегруженный оператор.

Классы и возможные действия над классами

|  |  |  |
| --- | --- | --- |
| Вариант | Класс | Возможности класса |
| 11 | Матрица целых чисел | Умножение матрицы на вещественное число и присвоение результата третьей переменной-матрице осуществляется одной строкой С = А\*b, где b - целое число |

***Код программы***

#include <iostream>

**using** **namespace** std;

// Объект класса Matrix хранит в себе целочисленную матрицу указанного размера.// Эту матрицу можно вывести или умножить на вещественное число, причем// в последнем случае результат запишется в новую матрицу.

**class** Matrix

{

**private**:

**int** size\_i;

**int** size\_j;

**int**\*\* matrix;

**public**:

Matrix() // Конструктор по умолчанию

{

size\_i = 3;

size\_j = 3;

matrix = **new** **int**\*[size\_i];

**for** (**int** i = 0; i < size\_i; i++)

{

matrix[i] = **new** **int**[size\_j];

}

}

Matrix(**int** s\_i, **int** s\_j) // Параметризованный конструктор

{

size\_i = s\_i;

size\_j = s\_j;

matrix = **new** **int**\*[size\_i];

**for** (**int** i = 0; i < size\_i; i++)

{

matrix[i] = **new** **int**[size\_j];

}

}

Matrix(**const** Matrix& mtrx) // Конструктор копирования

{

size\_i = mtrx.size\_i;

size\_j = mtrx.size\_j;

matrix = **new** **int**\*[size\_i];

**for** (**int** i = 0; i < size\_i; i++)

{

**for** (**int** j = 0; j < size\_j; j++)

{

matrix[i][j] = mtrx.matrix[i][j];

}

}

}

~Matrix() // Деструктор, освобождающий выделенную для объекта класса память

{

**for** (**int** i = 0; i < size\_i; i++)

{

**delete**[] matrix[i];

}

**delete** [] matrix;

}

**void** fill() // Заполняет матрицу числами от 1 до j (число столбцов)

{

**for** (**int** i = 0; i < size\_i; i++)

{

**for** (**int** j = 0; j < size\_j; j++)

{

matrix[i][j] = j + 1;

}

}

}

**int** get(**int** i, **int** j) // Возвращает значение [i][j]-элемента

{

**return** matrix[i][j];

}

**void** set(**int** val, **int** i, **int** j) // Устанавливает в [i][j]-элементу значение val

{

matrix[i][j] = val;

}

**friend** ostream& **operator** <<(ostream& out, Matrix& M) // Перегрузка оператора для вывода матрицы

{

**for** (**int** i = 0; i < M.size\_i; i++)

{

**for** (**int** j = 0; j < M.size\_j; j++)

{

out << M.get(i, j) << ' ';

}

out << '\n';

}

**return** out;

}

Matrix **operator** \*(**int** num) // Перегрузка оператора для умножения матрицы на число

{

**for** (**int** i = 0; i < **this**->size\_i; i++)

{

**for** (**int** j = 0; j < **this**->size\_j; j++)

{

**this**->set(num, i, j);

}

}

**return** \***this**;

}

};

**int** main()

{

Matrix A = Matrix();

A.fill();

cout << "Matrix A:\n" << A << '\n';

Matrix B = Matrix(3, 4);

B.fill();

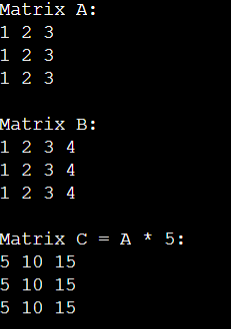
cout << "Matrix B:\n" << B << '\n';

Matrix C = A \* 5;

cout << "Matrix C = A \* 5:\n" << C << '\n';

}

***Пример работы программы***



***Вывод:***

В ходе выполнения работы изучил виды и принципы использования конструкторов, назначение деструктора, способы перегрузки операторов и реализовал на языке С++ класс, который способен хранить матрицу, умножать её на число и выводить результат (с использованием перегруженных операторов \* и <<).

***Контрольные вопросы***

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

*Конструктор -* специальная функция (метод) класса, срабатывающая в момент создания объекта класса. Имя конструктора совпадает с именем класса и ничего не возвращает (при этом также не является void).

*Деструктор -* специальная функция (метод) класса, срабатывающая в момент удаления объекта класса. Имя деструктора совпадает с именем класса, но в начале также имеет символ «~». В деструкторе освобождается память, выделенная во время создания и работы объекта класса.

1. **Какие бывают конструкторы?**

*Конструктор по умолчанию* - без указания каких-либо параметров.

*Параметризованный конструктор -* конструктор, принимающий некоторые параметры.

*Конструктор копирования -* копирует некоторые данные из одного объекта класса в другой.

*Конструктор перемещения (присваивания) -* вызывает обмен данными между двумя объектами класса (подобен функции swap()для обмена значениями двух переменных)

1. **Зачем нужен конструктор копирования?**

Конструктор копирования нужен, чтобы безопасно создать копию объекта класса. Также он необходим в классах, где происходит динамическое выделение памяти (имеется указатель, для которого память выделяется динамически с помощью оператора new или других функций).

Конструктор копирования может пригодиться в следующих ситуациях:

- Чтобы передать объект в функцию как параметр по значению, а не по ссылке;

- Чтобы вернуть объект как результат работы некоторой функции;

- Чтобы создать ещё один объект класса с теми же свойствами, что и у первого.

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

Часто оператор присваивания (=) перегружается для «копирования» значений из одного объекта в другой путём их присваивания (obj1 *=* obj2). Это позволяет сократить количество кода, а следовательно, сократить число потенциальных ошибок.

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

Обычно это либо глобальные функции, являющиеся дружественными для класса, либо функции, определяемые внутри этого класса.

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

Когда происходит присваивание одного объекта другому. Например, пусть определены класс Table и объект класса - table1. Тогда конструктор копирования сработает при создании ещё одного объекта и присвоении ему объекта table1:

Table table2 = table1;

1. **Когда вызывается деструктор класса?**

При выходе объекта класса из области видимости программы (завершение выполнения программы, когда освобождается выделенная для её выполнения память) либо при явном удалении объекта оператором delete*.*