**ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНТЕЛЕКТУАЛЬНИХ ТЕХНОЛОГІЙ І ЗВ’ЯЗКУ**

**Звіт**

**з дисципліни Проектний Практикум**

**Лабораторна робота №5**

**на тему: «Проектування і реалізація програми з перевантаженням функцій»»**

Виконав: студент групи ІПЗ-3.04

Бухта М.М

       \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Перевірив: Багачук Д.Г.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Одеса  2023**

**МЕТА РОБОТИ**

Засвоєння поняття статичного поліморфізму через перевантаження функцій; набуття навичок використання практичних прийомів перевантаження функцій та аргументів за замовчуванням.

**ЗАВДАННЯ**

**Опис завдання:**

Порядок виконання роботи:

1. Для реалізації програмного коду створити два класи: матриця і вектор. У кожному з класів задати конструктори і декструктори.
2. Як закриті елементи класів, створити динамічні масиви, що зберігаються відповідно матрицю або вектор та їх розмірність. Динамічний масив матриці має бути двовимірним, вектора – одновимірним.
3. Додати дружні функції множення вектора на матрицю і навпаки для цих двох класів. Як параметри передати об’єкти створених класів.
4. Перевантажити створену функцію для множення вектора на число і навпаки.
5. Перевантажити створену функцію для множення матриці на число і навпаки.
6. Перевантажити створену функцію для множення двох матриці (як об’єктів класу матриця).
7. Перевантажити створену функцію для множення двох векторів (як об’єктів класу вектор).
8. Під час реалізації перевантажених функцій перевірити можливість такого множення (не всі вектори і матриці можна перемножати).
9. В основній частині програми продемонструвати роботу перевантажених функцій.

**Код програми:**

**main.cpp**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\* Laboratory work #5;              \**

*\* Student Bukhta Mykyta;           \**

*\* Grade: 3;                        \**

*\* Group Software Engineering 3.04; \*                       \**

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\*/*

#include "Vector"

#include "Matrix"

#include <iostream>

using namespace lab\_5;

void print(const Vector<int32\_t> &vector) {

    uint64\_t size = vector.size();

    for (uint64\_t i{0}; i < size; ++i) {

        std::cout << vector[i] << ", ";

    }

    std::cout << std::endl;

}

void print(const Matrix<int32\_t> &matrix) {

    uint64\_t size = matrix.rows\_count();

    for (uint64\_t i{0}; i < size; ++i) {

        print(matrix[i]);

    }

}

int main(int argc, char \*\*argv) {

    Vector<int32\_t> vector = {1, 2, 3};

    Matrix<int32\_t> matrix = {

        {1, 2, 3},

        {4, 5, 6},

        {7, 8, 9}

    };

    std::cout << "3 \* vector \* 3" << std::endl;

    print(3 \* vector \* 3);

    std::cout << "\n3 \* matrix \* 3" << std::endl;

    print(3 \* matrix \* 3);

    std::cout << "\nvector \* vector" << std::endl;

    print(vector \* vector);

    std::cout << "\nmatrix \* matrix" << std::endl;

    print(matrix \* matrix);

    std::cout << "\vector \* matrix \* vector" << std::endl;

    print(vector \* matrix \* vector);

    return 0;

}

**Vector.hpp**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\* Laboratory work #5;              \**

*\* Student Bukhta Mykyta;           \**

*\* Grade: 3;                        \**

*\* Group Software Engineering 3.04; \*                       \**

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\*/*

#ifndef BUKHTAMYKYTA\_LAB\_5\_VECTOR\_HPP

#define BUKHTAMYKYTA\_LAB\_5\_VECTOR\_HPP

#include <inttypes.h>

#include <initializer\_list>

namespace lab\_5 {

#define \_\_COMMON\_VECTOR\_LIST\_SIZE\_\_ 10

template <typename T>

class Vector {

public:

    Vector(void);

    Vector(const std::initializer\_list<T> &init\_list);

    Vector(const Vector<T> &other);

    Vector(Vector<T> &&other);

    virtual ~Vector(void);

    void operator= (const std::initializer\_list<T> &init\_list);

    void operator= (const Vector<T> &other);

    void operator= (Vector<T> &&other);

    T& operator[] (uint64\_t index) const noexcept;

*/\* Description:*

*\* Multiple vectors to each other.*

*\**

*\* Return values:*

*\* Return vector with min vector size between two vector;*

*\*/*

    Vector<T> operator\* (const Vector<T> &other) const;

    Vector<T> operator\* (int32\_t val) const;

    template <typename U> *// to remove the warning during compilation.*

    friend Vector<U> operator\* (int32\_t val, const Vector<U> &other);

    uint64\_t size(void) const noexcept;

    void resize(uint64\_t size);

    void clear(void);

private:

    T \*m\_dynamic\_array = nullptr;

    uint64\_t m\_size;

    uint64\_t m\_inited\_size;

};

} *// !lab\_5;*

#endif *// !BUKHTAMYKYTA\_LAB\_5\_VECTOR\_HPP;*

**Vector.cpp**

#include "Vector.hpp"

#include <cstring>

#include <utility>

#include <algorithm>

namespace lab\_5 {

template <typename T>

Vector<T>::Vector(void)

{

    m\_size = 0;

    m\_inited\_size = \_\_COMMON\_VECTOR\_LIST\_SIZE\_\_;

    m\_dynamic\_array = new T[m\_inited\_size];

}

template <typename T>

Vector<T>::Vector(const std::initializer\_list<T> &init\_list) {

    operator=(init\_list);

}

template <typename T>

Vector<T>::Vector(const Vector<T> &other) {

    operator=(other);

}

template <typename T>

Vector<T>::Vector(Vector<T> &&other) {

    operator=(std::move(other));

}

template <typename T>

Vector<T>::~Vector(void) {

    clear();

}

template <typename T>

void Vector<T>::operator= (const std::initializer\_list<T> &init\_list) {

    clear();

    m\_size = init\_list.size();

    m\_inited\_size = m\_size + \_\_COMMON\_VECTOR\_LIST\_SIZE\_\_;

    m\_dynamic\_array = new T[m\_inited\_size];

    uint64\_t i{0};

    for (auto init\_elem : init\_list) {

        m\_dynamic\_array[i++] = init\_elem;

    }

}

template <typename T>

void Vector<T>::operator= (const Vector<T> &other) {

    clear();

    this->m\_size = other.m\_size;

    this->m\_inited\_size = this->m\_size + \_\_COMMON\_VECTOR\_LIST\_SIZE\_\_;

    this->m\_dynamic\_array = new T[this->m\_inited\_size];

    for (uint64\_t i{0}; i < this->m\_size; ++i) {

        this->m\_dynamic\_array[i] = other.m\_dynamic\_array[i];

    }

}

template <typename T>

void Vector<T>::operator= (Vector<T> &&other) {

    this->m\_dynamic\_array = other.m\_dynamic\_array;

    this->m\_inited\_size = other.m\_inited\_size;

    this->m\_size = other.m\_size;

    other.m\_dynamic\_array = nullptr;

    other.m\_inited\_size = 0;

    other.m\_size = 0;

}

template <typename T>

T& Vector<T>::operator[] (uint64\_t index) const noexcept {

    return m\_dynamic\_array[index];

}

template <typename T>

Vector<T> Vector<T>::operator\* (const Vector<T> &other) const {

    uint64\_t ret\_vector\_size = std::min(this->size(), other.size());

    Vector<T> ret;

    ret.resize(ret\_vector\_size);

    for (uint64\_t i{0}; i < ret\_vector\_size; ++i) {

        ret[i] = this->m\_dynamic\_array[i] \* other.m\_dynamic\_array[i];

    }

    return std::move(ret);

}

template <typename T>

Vector<T> Vector<T>::operator\* (int32\_t val) const {

    Vector<T> ret{\*this};

    for (uint64\_t i{0}; i < ret.m\_size; ++i) {

        ret[i] = this->m\_dynamic\_array[i] \* val;

    }

    return std::move(ret);

}

template <typename U>

Vector<U> operator\* (int32\_t val, const Vector<U> &other) {

    return std::move(other.operator\*(val));

}

template <typename T>

uint64\_t Vector<T>::size(void) const noexcept {

    return m\_size;

}

template <typename T>

void Vector<T>::resize(uint64\_t size) {

    if (m\_size == size) {

        return;

    }

    T \*new\_dynamic\_array = new T[size];

    uint64\_t min\_size = std::min(size, m\_size);

    uint64\_t max\_size = std::max(size, m\_size);

    uint64\_t i{0};

    for (; i < min\_size; ++i) {

        new\_dynamic\_array[i] = m\_dynamic\_array[i];

    }

    for (; i < max\_size; ++i) {

        new\_dynamic\_array[i] = T{};

    }

    clear();

    m\_dynamic\_array = new\_dynamic\_array;

    m\_inited\_size = m\_size = size;

}

template <typename T>

void Vector<T>::clear(void) {

    if (m\_dynamic\_array) {

        delete[] m\_dynamic\_array;

        m\_dynamic\_array = nullptr;

    }

}

} *// !lab\_5;*

**Vector**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\* Laboratory work #5;              \**

*\* Student Bukhta Mykyta;           \**

*\* Grade: 3;                        \**

*\* Group Software Engineering 3.04; \*                       \**

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\*/*

#ifndef BUKHTAMYKYTA\_LAB\_5\_VECTOR

#define BUKHTAMYKYTA\_LAB\_5\_VECTOR

#include "Vector.hpp"

#include "Vector.cpp"

#endif *// !BUKHTAMYKYTA\_LAB\_5\_VECTOR;*

**Matrix.hpp**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\* Laboratory work #5;              \**

*\* Student Bukhta Mykyta;           \**

*\* Grade: 3;                        \**

*\* Group Software Engineering 3.04; \*                       \**

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\*/*

#ifndef BUKHTAMYKYTA\_LAB\_5\_MATRIX\_HPP

#define BUKHTAMYKYTA\_LAB\_5\_MATRIX\_HPP

#include "Vector"

#include <inttypes.h>

#include <initializer\_list>

namespace lab\_5 {

template <typename T>

class Matrix {

public:

    Matrix(void);

    Matrix(const std::initializer\_list<std::initializer\_list<T>> &init\_matrix);

    Matrix(const Matrix<T> &other);

    Matrix(Matrix<T> &&other);

    Matrix(const Vector<Vector<T>> &matrix);

    virtual ~Matrix(void) = default;

    void operator= (const std::initializer\_list<std::initializer\_list<T>> &init\_matrix);

    void operator= (const Matrix<T> &other);

    void operator= (Matrix<T> &&other);

    void operator= (const Vector<Vector<T>> &matrix);

    Vector<T>& operator[] (uint64\_t index) const noexcept;

*/\* Description:*

*\*  Multiple matrix to each other;*

*\**

*\* Return values:*

*\*  If count of colums of the first matrix is*

*\* not equals to count of rows of the second matrix*

*\* (arg other), !!! the empty matrix should be returned !!!;*

*\**

*\*/*

    Matrix<T> operator\* (const Matrix<T> &other) const;

    Matrix<T> operator\* (const Vector<T> &vector) const;

    template <typename U> *// to remove the warning during compilation;*

    friend Matrix<U> operator\* (const Vector<U> &vector, const Matrix<U> &matrix);

    Matrix<T> operator\* (int32\_t val) const;

    template <typename U> *// to remove the warning during compilation;*

    friend Matrix<U> operator\* (int32\_t val, const Matrix<U> &other);

    uint64\_t rows\_count(void) const noexcept;

    uint64\_t colums\_count(void) const noexcept;

    void resize\_rows(uint64\_t size);

    void resize\_colums(uint64\_t size);

private:

    Vector<Vector<T>> m\_matrix;

    uint64\_t m\_colums\_count;

};

} *// !lab\_5;*

#endif *// !BUKHTAMYKYTA\_LAB\_5\_MATRIX\_HPP;*

**Matrix.cpp**

#include "Matrix.hpp"

#include <algorithm>

namespace lab\_5 {

template <typename T>

Matrix<T>::Matrix(void)

    : m\_colums\_count{0}

{

}

template <typename T>

Matrix<T>::Matrix(const std::initializer\_list<std::initializer\_list<T>> &init\_matrix) {

    operator=(init\_matrix);

}

template <typename T>

Matrix<T>::Matrix(const Matrix<T> &other) {

    operator=(other);

}

template <typename T>

Matrix<T>::Matrix(Matrix<T> &&other) {

    operator=(std::move(other));

}

template <typename T>

Matrix<T>::Matrix(const Vector<Vector<T>> &matrix) {

    operator=(matrix);

}

template <typename T>

void Matrix<T>::operator= (const std::initializer\_list<std::initializer\_list<T>> &init\_matrix) {

    m\_colums\_count = 0;

    m\_matrix.clear();

    m\_matrix.resize(init\_matrix.size());

    uint64\_t i{0};

    for (auto init\_list : init\_matrix) {

        m\_colums\_count = std::max(init\_list.size(), m\_colums\_count);

        m\_matrix[i++] = init\_list;

    }

}

template <typename T>

void Matrix<T>::operator= (const Matrix<T> &other) {

    this->m\_matrix = other.m\_matrix;

    this->m\_colums\_count = other.m\_colums\_count;

}

template <typename T>

void Matrix<T>::operator= (Matrix<T> &&other) {

    this->m\_colums\_count = other.m\_colums\_count;

    this->m\_matrix = std::move(other.m\_matrix);

    other.m\_colums\_count = 0;

}

template <typename T>

void Matrix<T>::operator= (const Vector<Vector<T>> &matrix) {

    m\_colums\_count = 0;

    m\_matrix = matrix;

    uint64\_t matrix\_size = matrix.size();

    for (uint64\_t i{0}; i < matrix\_size; ++i) {

        m\_colums\_count = std::max(m\_colums\_count, matrix[i].size());

    }

}

template <typename T>

Vector<T>& Matrix<T>::operator[] (uint64\_t index) const noexcept {

    return m\_matrix[index];

}

template <typename T>

Matrix<T> Matrix<T>::operator\* (const Matrix<T> &other) const {

    uint64\_t other\_rows\_count = other.rows\_count();

    if (m\_colums\_count != other\_rows\_count) {

        return {};

    }

    Matrix<T> ret;

    uint64\_t this\_rows\_count = rows\_count();

    ret.resize\_rows(this\_rows\_count);

    ret.resize\_colums(other.m\_colums\_count);

    T filling\_value{0};

    for (uint64\_t this\_i{0}; this\_i < this\_rows\_count; ++this\_i) {

        for (uint64\_t other\_j{0}; other\_j < other.m\_colums\_count; ++other\_j) {

*// m\_colums\_count == other\_rows\_cout, so it is a general (common) size of both matrix;*

            for (uint64\_t general\_index{0}; general\_index < m\_colums\_count; ++general\_index) {

                filling\_value += this->m\_matrix[this\_i][general\_index] \* other.m\_matrix[general\_index][other\_j];

            }

            ret[this\_i][other\_j] = filling\_value;

            filling\_value = 0;

        }

    }

    return std::move(ret);

}

template <typename T>

Matrix<T> Matrix<T>::operator\* (const Vector<T> &vector) const {

    if (this->m\_colums\_count != vector.size()) {

        return {};

    }

    Matrix<T> ret;

    ret.resize\_rows(rows\_count());

    ret.m\_colums\_count = this->m\_colums\_count;

    for (uint64\_t i{0}; i < this->m\_colums\_count; ++i) {

        ret[i] = std::move(this->m\_matrix[i] \* vector);

    }

    return std::move(ret);

}

template <typename U> *// to remove the warning during compilation;*

Matrix<U> operator\* (const Vector<U> &vector, const Matrix<U> &matrix) {

    return std::move(matrix.operator\*(vector));

}

template <typename T>

Matrix<T> Matrix<T>::operator\* (int32\_t val) const {

    return std::move(Matrix<T>{m\_matrix \* val});

}

template <typename U>

Matrix<U> operator\* (int32\_t val, const Matrix<U> &other) {

    return std::move(other.operator\*(val));

}

template <typename T>

uint64\_t Matrix<T>::rows\_count(void) const noexcept {

    return m\_matrix.size();

}

template <typename T>

uint64\_t Matrix<T>::colums\_count(void) const noexcept {

    return m\_colums\_count;

}

template <typename T>

void Matrix<T>::resize\_rows(uint64\_t size) {

    uint64\_t matrix\_size = m\_matrix.size();

    if (matrix\_size == size) {

        return;

    }

    m\_matrix.resize(size);

*// If new rows count > original size, we should to extend new lines;*

    if (matrix\_size < size) {

        for (uint64\_t i{matrix\_size}; i < size; ++i) {

            m\_matrix[i].resize(m\_colums\_count);

        }

    }

}

template <typename T>

void Matrix<T>::resize\_colums(uint64\_t size) {

    uint64\_t matrix\_size = m\_matrix.size();

    m\_colums\_count = size;

    for (uint64\_t i{0}; i < matrix\_size; ++i) {

        m\_matrix[i].resize(size);

    }

}

} *// !lab\_5;*

**Matrix**

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\* Laboratory work #5;              \**

*\* Student Bukhta Mykyta;           \**

*\* Grade: 3;                        \**

*\* Group Software Engineering 3.04; \*                       \**

*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\**

*\*/*

#ifndef BUKHTAMYKYTA\_LAB\_5\_MATRIX

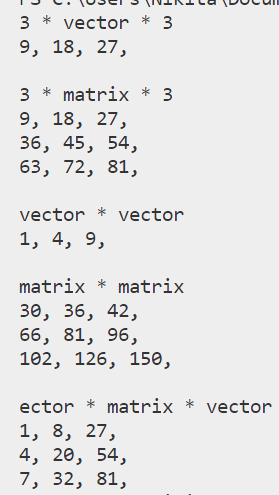
#define BUKHTAMYKYTA\_LAB\_5\_MATRIX

#include "Matrix.hpp"

#include "Matrix.cpp"

#endif *// !BUKHTAMYKYTA\_LAB\_5\_MATRIX;*

**Результат виконання:**



**ВИСНОВОК**

Після завершення цієї роботи я зрозумів, що статичний поліморфізм через перевантаження функцій є потужним інструментом в програмуванні. Я засвоїв ідею, що можна мати функції з однаковим ім'ям, але з різними параметрами, і це дозволяє створювати більш універсальний та зрозумілий код.