**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**

```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);
}
```

```cpp
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;
}
```

```cpp
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

```cpp
/************************************
 * 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}
{

}
```

```cpp
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:**

```
3 * vector * 3
9, 18, 27,

3 * matrix * 3
9, 18, 27,
36, 45, 54,
63, 72, 81,

vector * vector
1, 4, 9,

matrix * matrix
30, 36, 42,
66, 81, 96,
102, 126, 150,

ector * matrix * vector
1, 8, 27,
4, 20, 54,
7, 32, 81,
```

# 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.